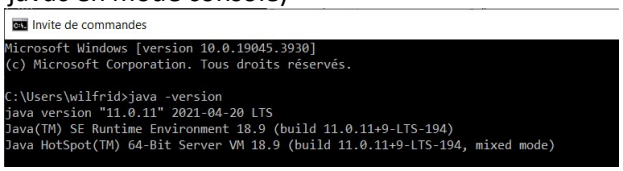
Création des librairies (dll) et wrappers Java et Python pour la gestion du fichier de mapping de télémétrie du Drone Char

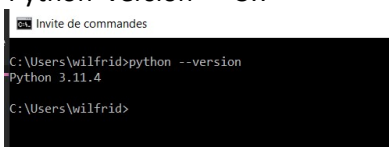
1- Prérequis :

Logiciels installés,

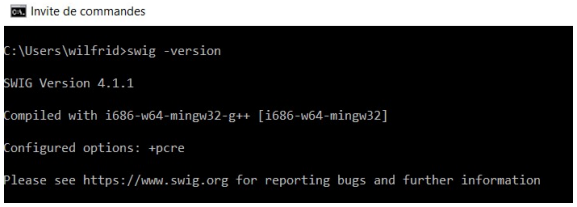
- Java version JDK 11 avec les variables d'environnement correctement initialisées (tester l'accès à java et javac en mode console)



- Python version >= 3.7



- Swig version >= 4.0.2 pour la génération des wrappers Java et Python



https://www.swig.org/download.html

https://sourceforge.net/projects/swig/files/swigwin/swigwin-4.2.0/swigwin-4.2.0.zip/download?use_mirror=netix

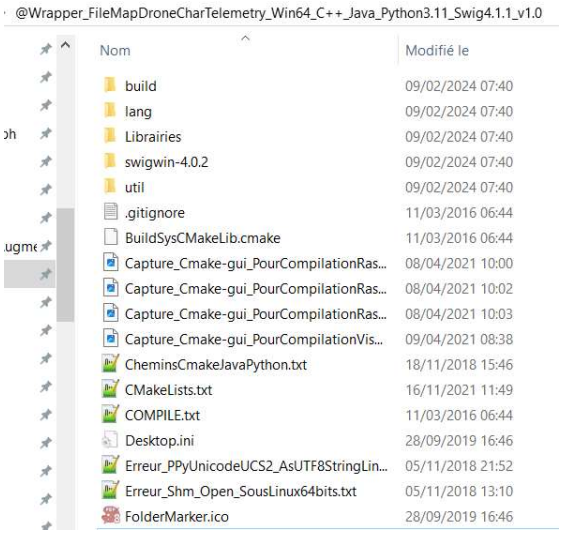
- Cmake-gui version >=3.0

https://cmake.org/download/

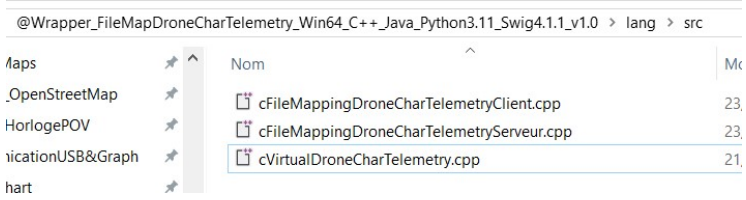
Binary distributions:

Platform	Files
Windows x64 Installer:	cmake-3.28.3-windows-x86_64.msi

2- Structure du package Cmake :



- Les fichiers sources .cpp



- Les fichiers d'entêtes .h

- Les fichiers .swig de génération des Wrappers Java et Python

  Ce sont des fichiers pratiquement identiques aux fichiers d'entêtes .h mais avec des directives permettant la conversion des types de données entre C++ et Java ou C++ et Python.

  Les fichiers concernant les wrappers Java sont préfixés de « jlib ».
  Les fichiers concernant les wrappers Python sont préfixés de « pylib »

3- Adapter le package DroneTelemetry pour créer le package DroneCharTelemetry :

- La structure de données du DroneCharTelemetry sera la suivante,

```
bool           MutexBlocAccess;          // Mutex to protect ressouce access
int            BatteryValue;             // Drone Battery Value
char           DriveTime[30];            // Drone Drive Time
char           TempC[30];                // Drone Temp Celsius
char           TempF[30];                // Drone Temp Farenheit
char           Altitude[30];             // Drone Altitude
double         Ax;                       // Drone acceleration X
double         Ay;                       // Drone acceleration Y
double         Az;                       // Drone acceleration Z
double         FrontDistance;            // Drone FrontDistance
double         BackDistance;             // Drone BackDistance
double         Pressure;                 // Drone Pressure
unsigned char* TelloEduTelemetryDataPtr; // Pointer on TelloEduTelemetry data buffer
```

- Fichiers C++ à modifier :
  cFileMappingDroneCharTelemetryClient.cpp
  cFileMappingDroneCharTelemetryServeur.cpp
  cVirtualDroneCharTelemetry.cpp

  cFileMappingDroneCharTelemetryClient.h
  cFileMappingDroneCharTelemetryServeur.h
  cVirtualDroneCharTelemetry.h

- Fichiers Swig Wrapper Java à modifier :
  jlibFileMappingDroneCharTelemetryClient_swig.i
  jlibFileMappingDroneCharTelemetryServeur_swig.i
  jlibVirtualDroneCharTelemetry_swig.i

- Fichiers Swig Wrapper Python à modifier :
  pylibFileMappingDroneCharTelemetryClient_swig.i
  pylibFileMappingDroneCharTelemetryServeur_swig.i
  pylibVirtualDroneCharTelemetry_swig.i

4- Configuration du projet cmake :

La configuration s'effectue dans le fichier
@Wrapper_FileMapDroneCharTelemetry_Win64_C++_Java_Python3.11_Swig4.1.1_v1.0\lang\swig\CMakeLists.txt

Ce fichier contient les directives de compilation pour générer les différentes dll et les wrappers Java et Python.
Faire attention à « la casse » (minuscules et majuscules) !!!!!!

- Remplacer les noms de variables
  FILEMAPPINGTELLOEDUTELEMETRY par FILEMAPPINGDRONECHARTELEMETRY

- Remplacer les noms
  TelloEdu par DroneChar

5- Configuration de la chaine de compilation cmake :

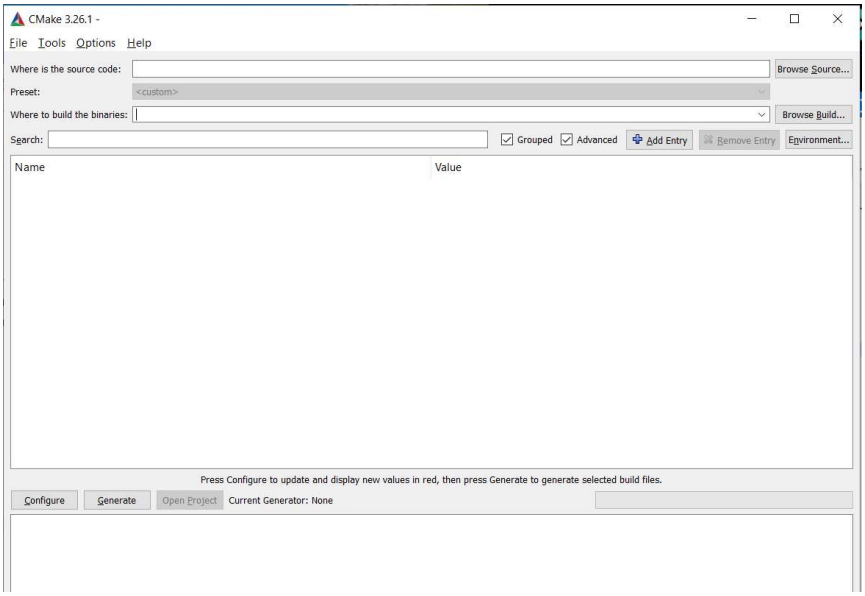
Une fois que le contenu des différents fichiers C++ et swig est complété, nous allons procéder à la configuration de la chaine de compilation cmake avec l'outil cmake-gui.
Cette phase nous permettra de générer automatiquement les fichiers de projet VisualStudio pour créer les DLL C++ (.dll .lib .pyd) et les Wrappers Java (.java) et Python (.py)

- Modifier le nom du projet dans le fichier @Wrapper_FileMapDroneCharTelemetry_Win64_C++_Java_Python3.11_Swig4.1.1_v1.0/ CMakeLists.txt

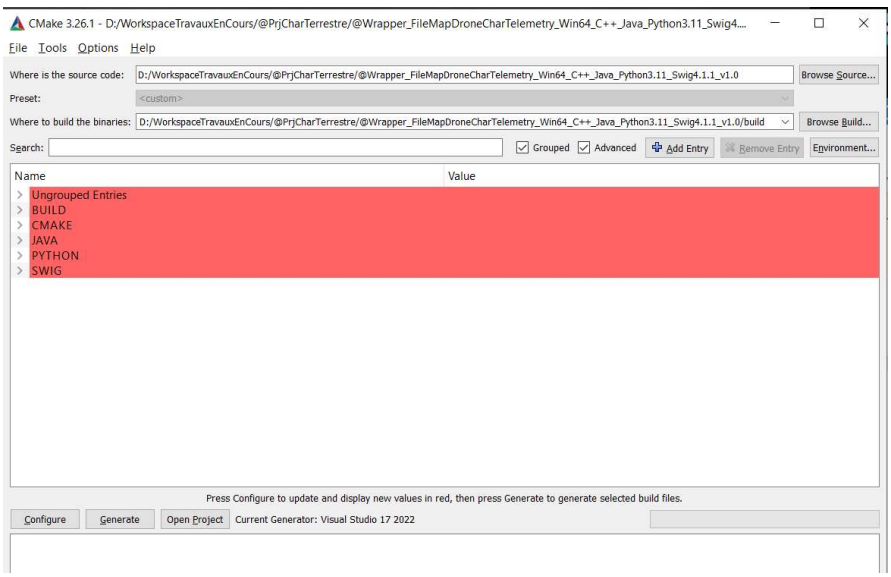
```
##############################################################################
# Project declaration and options
##############################################################################

#Project declaration

PROJECT(FILEMAPDRONECHARTELEMETRY)
INITIALIZE_BUILD()

#Common Options
```

Ici "FILEMAPDRONECHARTELEMETRY"

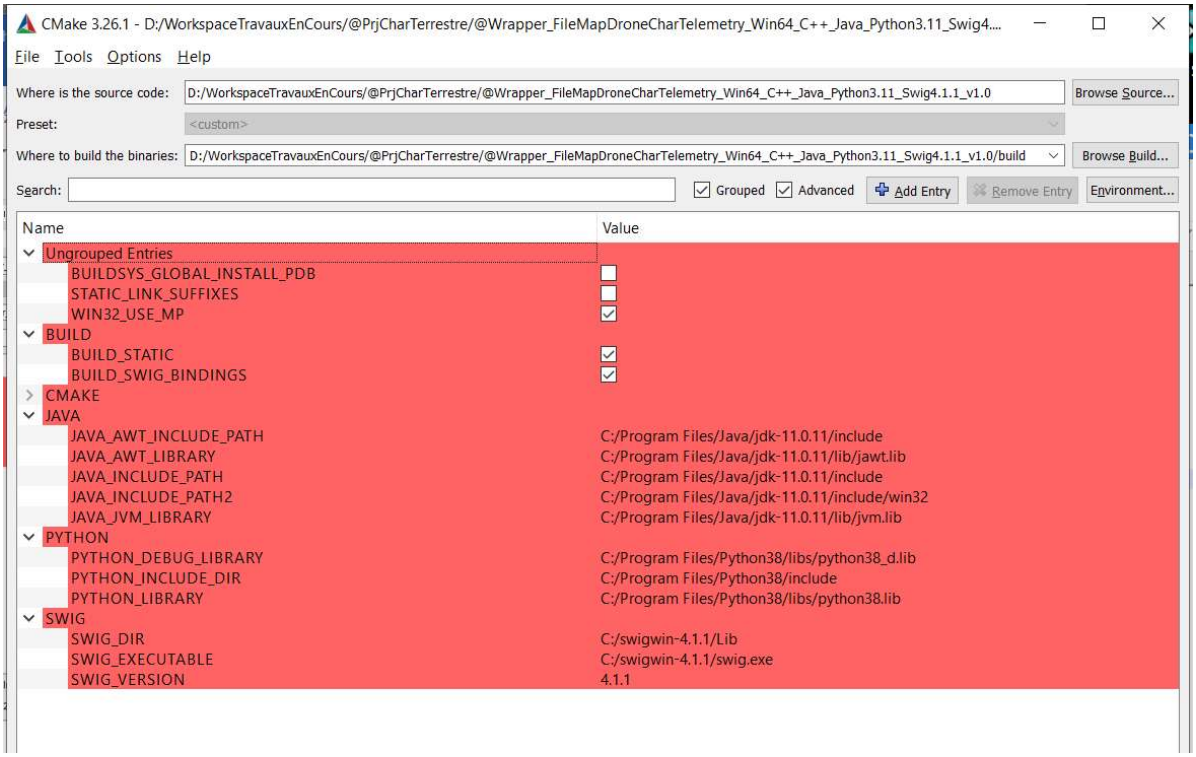
- Lancer le logiciel Cmake-gui



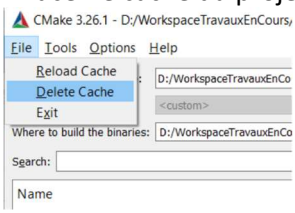
- Utiliser les boutons « Browse Source » et « Browse Build » pour pointer sur votre projet.



- Vérifier que les éléments suivants sont bien configurés avec les versions installées sur le poste.

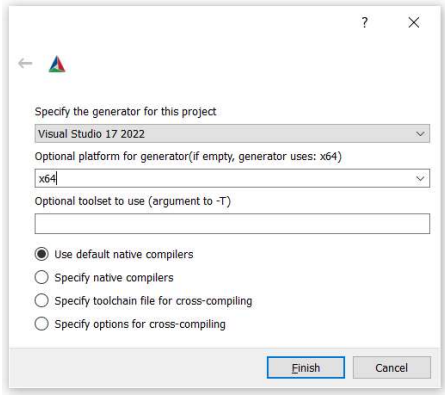


- Effacer le cache du projet.



- Configurer le projet pour la compilation en cliquant sur le bouton configure dans la zone inférieure de cmake-gui.vi

- Configurer le compilateur et le processeur utilisé.

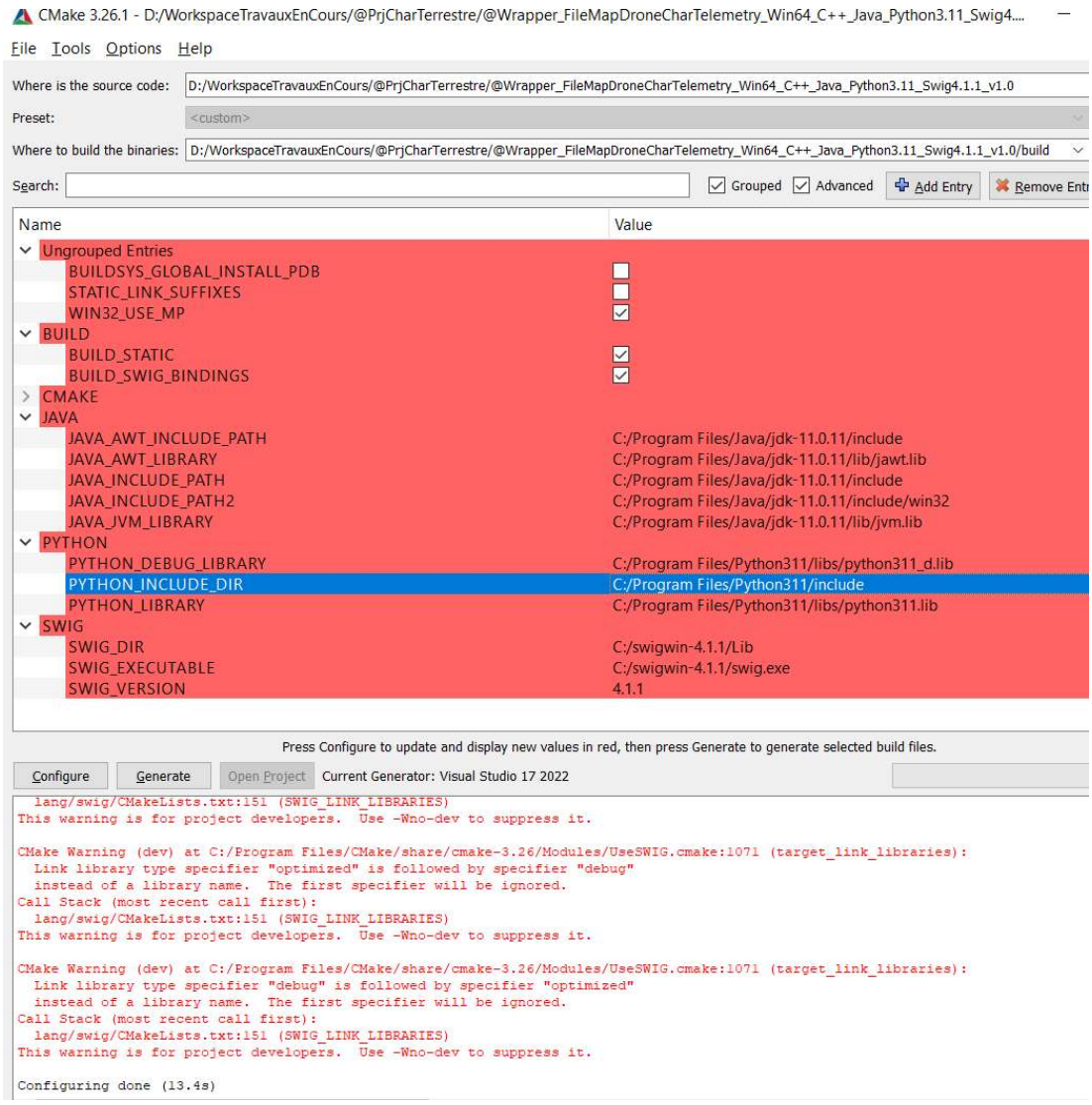


Suivant la version de Visual Studio installée utiliser la version la plus récente qui apparait dans la liste (ici VisualStudio 2022)

Compilation 64bits -> x64

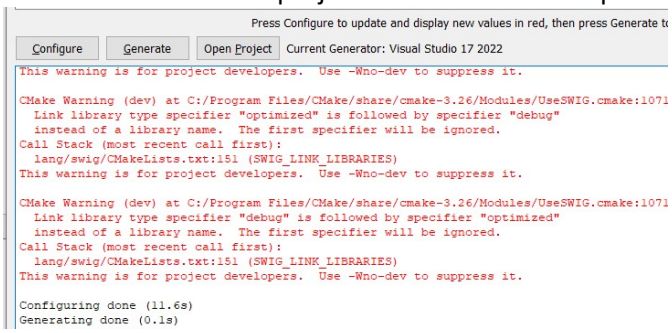
Utilisation du compilateur natif

- Vérifier que la configuration c'est bien passée



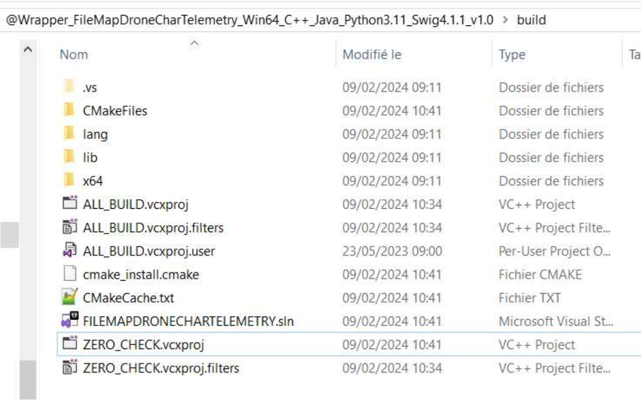
Ne pas tenir compte des warning en rouge, la configuration est correcte si vous obtenez le message « Configuring done (xxx s)
Revérifier que les paramètres dans la zone supérieure sont toujours corrects.

- Générer les fichiers de projets Visual Studio en cliquant sur le bouton « Generate » dans la zone inférieure.

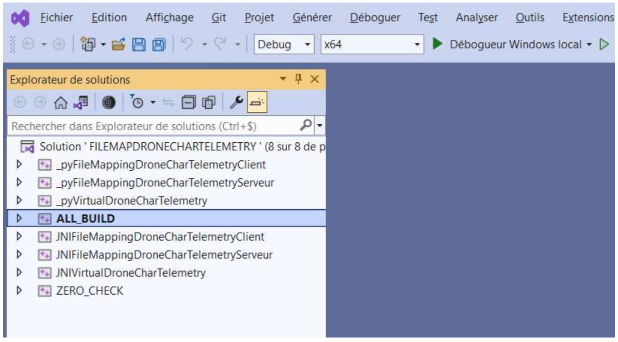


Ok si le message Generating done (xxx s) apparait

- Le fichier de projet Visual Studio « FILEMAPDRONECHARTELEMETRY.sln » doit être présent dans le dossier build.

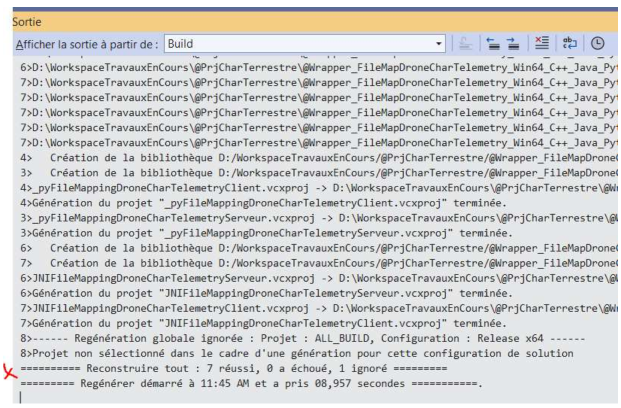


- Utiliser le bouton « Open Project » de Cmake-gui ou Double cliquer sur le fichier solution de projet « FILEMAPDRONECHARTELEMETRY.sln »



- Changer la génération des DLL en mode debug par le mode release

- Générer la solution



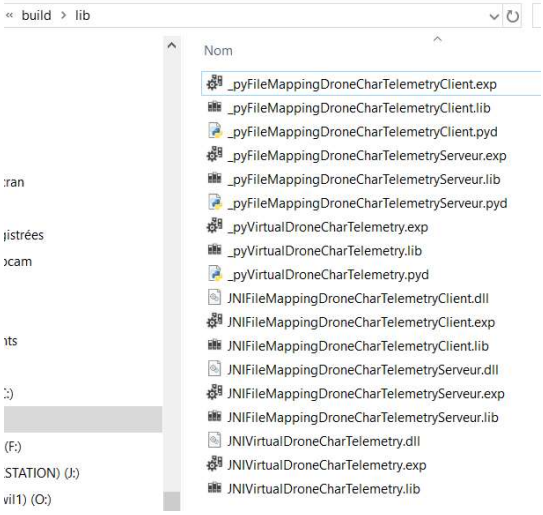
Vous devez obtenir 7 réussi et 0 échoué

- En cas d'erreur, faire une recherche dans le rapport de sortie de compilation (ci-dessus) sur le mot « error » Les lignes où se trouvent les erreurs vous mèneront au fichier source ayant causé l'erreur en faisant un double clic sur la ligne d'erreur.

  Il faudra après avoir corrigé les erreurs dans les fichiers sources refaire les étapes suivantes.
  1) Fermer Visual Studio
  2) Cliquer sur le bouton « Configure » de Cmake-gui
  3) Cliquer sur le bouton « Generate » de Cmake-gui
  4) Cliquer sur le bouton « Open Project » de Cmake-gui
  5) Nettoyer la solution, dans le menu Générer de Visual Studio
  6) Régénérer la solution, dans le menu Générer de Visual Studio

6- Récupération des librairies générées :

- Dans le dossier build/lib vous obtenez la liste de fichiers suivante :

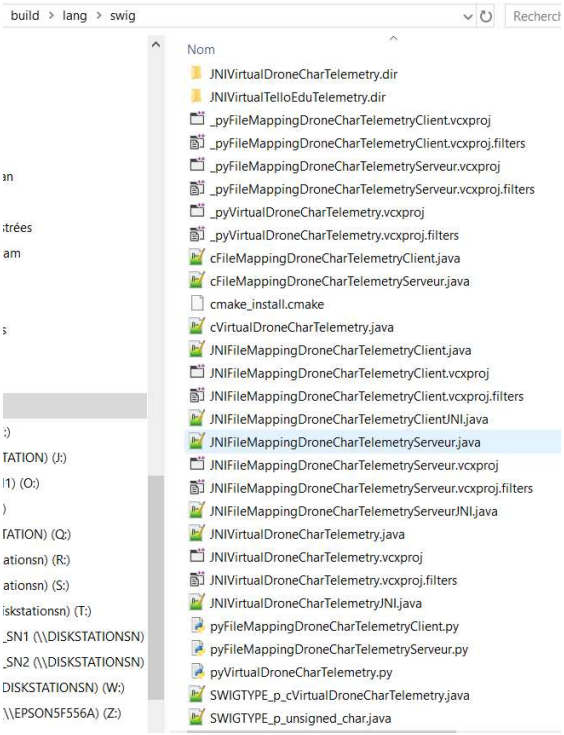


- Seuls les fichiers .dll nous seront utiles pour Java, les .pyd pour Python.

- Nous n'utiliserons pas non plus les librairies static .lib

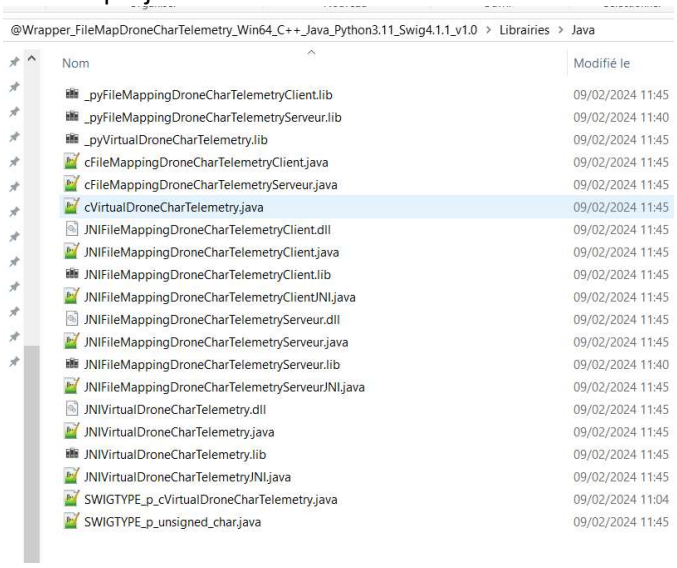
7- Récupération des Wrappers générés :

- Dans le dossier build/lang/swig vous obtenez la liste de fichiers suivante :

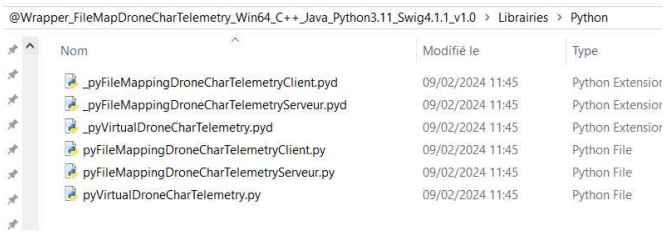


- Seuls les fichiers .java et .py seront utiles suivant le projet qui les utilisera.

- Pour un projet Java



- Pour un projet Python



8- Utilisation des librairies dans un projet Java :

Le projet Java il faudra mettre les .java dans le dossier src,

Créer un dossier Libs pour les dll

Exemple d'utilisation :

```
public class ThreadDroneCharTelemetryToFMP extends Thread
{
    static
    {
        try
        {
            if(System.getProperty("os.name").contains("Windows"))
            {
                    //System.loadLibrary("Libs/FileMapDroneCharTelemetry/JNIFileMappingDroneCharTelemetryClient");
                    System.loadLibrary("Libs/FileMapDroneCharTelemetry/JNIFileMappingDroneCharTelemetryServeur");
                    System.loadLibrary("Libs/FileMapDroneCharTelemetry/JNIVirtualDroneCharTelemetry");
            }
            else
            {
                    //System.load(ThreadPictureToFMP.class.getProtectionDomain().getCodeSource().getLocation().toURI().getPath().replace("
                    System.load(ThreadPictureToFMP.class.getProtectionDomain().getCodeSource().getLocation().toURI().getPath().replace("Te
                    System.load(ThreadPictureToFMP.class.getProtectionDomain().getCodeSource().getLocation().toURI().getPath().replace("Te
            }
            //System.out.println("Serveur Telemetrie librairies chargees.");
            //new WarningMsg("Serveur Telemetrie librairies\nChargees.", WarningMsg.TypeMsg.Msg_Info, true, 1, false);
        }
        catch(UnsatisfiedLinkError | URISyntaxException e)
        {
            System.out.println("Serveur Telemetrie librairies, impossible de charger les librairies.");
            new WarningMsg("Serveur Telemetrie librairies\nImpossible de charger les librairies.", WarningMsg.TypeMsg.Msg_Erreur, false, t
        }
    }
```

```
public ThreadDroneTelemetryToFMP(IHMDrone _IHM, int _index)
{
    this.monIHM = _IHM;
    this.index = _index;
    this.monServeurDroneCharTelemetryFMP = new cFileMappingDroneCharTelemetryServeur(true);
    if(!this.monServeurDroneCharTelemetryFMP.OpenServer("DroneCharTelemetryFMP"))
    {
        System.out.println("Serveur Telemetrie FileMap impossible a demarrer !!!");
        new WarningMsg("Serveur Telemetrie FileMap\nimpossible a demarrer !!!", WarningMsg.TypeMsg.Msg_Erreur, false, true, 2, false);
    }
    /*else
    {
        new WarningMsg("Serveur Telemetrie FileMap\ndemarre.", WarningMsg.TypeMsg.Msg_Info, true, 2, false);
    }*/
}
```

```
public void run()
{
    System.out.println("Debut du Thread < ThreadDroneTelemetryToFMP >");
    while(!this.StopThread)
    {
        if(this.monIHM.getFlagStop())
        {
            this.close();
            break;
        }

        // Ecriture des donnees de telemetry dans le fichier de mapping pour une animation chai3d
        //System.out.println("Ax: " + this.monIHM.getAx() + "   Ay: " + this.monIHM.getAy() + "   Az: " + this.monIHM.getAz());
        this.setDroneCharTelemetryToFileMap(this.monIHM.getBattery(this.index), this.monIHM.getTempsVol(this.index),
                                            this.monIHM.getTemperature(this.index), this.monIHM.getAltitude(this.index),
                                            this.monIHM.getAx(this.index), this.monIHM.getAy(this.index), this.monIHM.getAz(this.index));
        new ThreadTempo(100);
    }
    this.CloseTelemetryClient();
    System.out.println("Fin du Thread < ThreadDroneTelemetryToFMP >");
}
```

```
private void CloseTelemetryClient()
{
    if(this.TestTelemetryFMP != null)
    {
        this.TestTelemetryFMP.CloseClient();
        new ThreadTempo(200);
    }
}

private void setDroneCharTelemetryToFileMap(int _battery, String _DriveTime, String _TempC, String TempF,
                                            String _Altitude, double _Ax, double _Ay, double _Az, double _FrontDistance, double _BackDistan
{
    System.out.println("battery:" + _battery + "  FlyTime:" + _FlyTime
                       + "   Temperature:" + _TempC + "°C  Temperature:" + _TempF + "°F  Altitude:" + _Altitude
                       + "   Ax:" + _Ax + "   Ay:" + _Ay + "   Az:" + _Az + "  Front Dist: + _FrontDistance + "   Back Dist:" + _BackDista
    this.monServeurDroneCharTelemetryFMP.setVirtualDroneCharTelemetryMutexBlocAccess(true);
    this.monServeurDroneCharTelemetryFMP.setVirtualDroneCharTelemetryBatteryValue(_battery);
    this.monServeurDroneCharTelemetryFMP.setVirtualDroneCharTelemetryFlyTime(_FlyTime);
    this.monServeurDroneCharTelemetryFMP.setVirtualDroneCharTelemetryTemp(_Temp);
    this.monServeurDroneCharTelemetryFMP.setVirtualDroneCharTelemetryAltitude(_Altitude);
    this.monServeurDroneCharTelemetryFMP.setVirtualDroneCharTelemetryAx(_Ax);
    this.monServeurDroneCharTelemetryFMP.setVirtualDroneCharTelemetryAy(_Ay);
    this.monServeurDroneCharTelemetryFMP.setVirtualDroneCharTelemetryAz(_Az);
    this.monServeurDroneCharTelemetryFMP.setVirtualDroneCharTelemetryFrontDistance(_FrontDistance);
    this.monServeurDroneCharTelemetryFMP.setVirtualDroneCharTelemetryBackDistance(_BackDistance);
    this.monServeurDroneCharTelemetryFMP.setVirtualDroneCharTelemetryPressure(_Pressure);
    this.monServeurDroneCharTelemetryFMP.setVirtualDroneCharTelemetryMutexBlocAccess(false);
}
```