INSTALLATION DE L'IDE ARDUINO V 1.0

INTRODUCTION

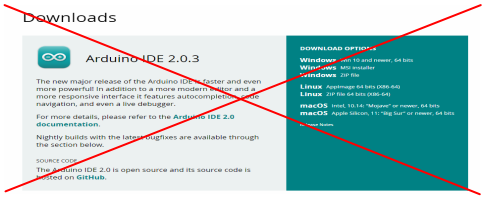
Cette notice a pour but de nous faire installer l'environnement "IDE Arduino" afin de nous donner accès facilement à cet écosystème très prisé dans le domaine radioamateur. Elle s'adresse principalement aux OM n'ayant aucune compétence en la matière afin de leur permettre de mener à bien certains projets basés sur les micro-contrôleurs. Elle complète l'article sur le "**BADGE STATIQUE E. PAPER POUR SALONS OM**" et elle va nous permettre, même aux non initiés au développement C et C++, de générer simplement le "binaire" (programme compilé) permettant d'utiliser et d'adapter les textes sur l'afficheur E. Paper.

Dans ce document seront abordés seulement les points concernant l'article sur le badge et l'installation de l'IDE sur une machine de type PC Windows®. Par contre, une fois l'IDE installé, il pourra être utilisé pour d'autres applications à base d'autres types de cartes de développement compatibles. Les étapes qui seront décrites sont les suivantes :

- Installation de l'IDE Arduino 8.19.
- Configuration de l'IDE.
- Installation des librairies spécifiques à la carte "ESP32" utilisée par le E. Paper.
- Configuration des paramètres dédiés à cette carte.
- Installation des librairies additives à l'IDE et utilisées par l'application finale.
- Mises à jour des diverses librairies.
- Informations sur les mises à jour automatiques.
- Compilation d'un programme de test.
- Conclusion.

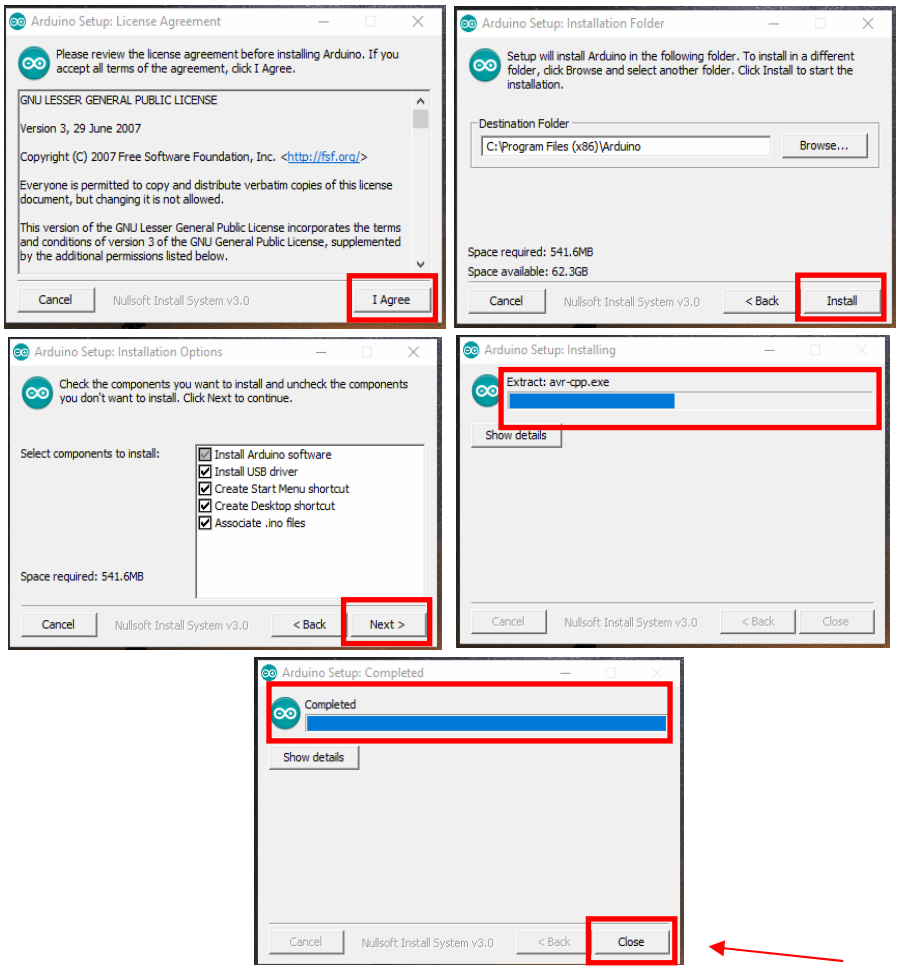
INSTALLATION DE L'IDE ARDUINO

Comme déjà expliqué dans l'article sur le badge E. Paper, il est nécessaire de télécharger cet environnement via le site : https://www.arduino.cc/en/software



La version 2.0.3 proposée par défaut est la dernière en date, mais malheureusement elle n'est pas encore compatible avec la carte utilisée par l'afficheur E. Paper. Il nous faut donc télécharger la version précédente :

Une fois la version **1.8.x** téléchargée, il nous suffit d'exécuter le programme d'installation et de suivre les étapes proposées. Les paramètres prédéfinis par défaut correspondent généralement à la plupart des utilisations :

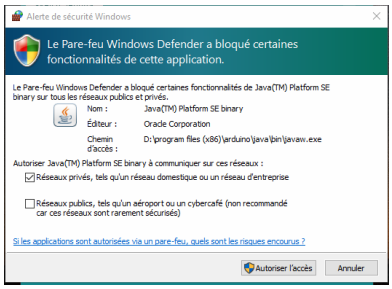


Pour finir l'installation, il suffit de sélectionner le bouton "Close"

CONFIGURATION DE L'IDE

Nous allons maintenant configurer l'IDE pour une utilisation simple et rapide. Normalement sur le "bureau" nous avons **un raccourci** pour ouvrir l'IDE. Un double-clic doit ouvrir un "sketch" vide par défaut.

Certaines fenêtres "pop-up" peuvent apparaître afin de demander l'autorisation d'accéder au réseau (et Internet) pour permettre les divers téléchargements automatiques et mises à jour. Il nous faut les valider sinon il nous sera impossible d'utiliser correctement cet IDE. Il n'y a aucun risque, ces demandes ne sont pas suspectes et elles sont normales, mais bloquées par le "pare-feu" de Windows®.



Exemple d'une fenêtre "Pop-up"

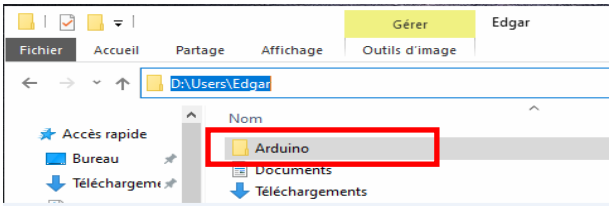
Ce "sketch" (projet de développement) ouvert par défaut ne nous servira à rien mais il nous faut le conserver à l'écran afin de garder l'IDE actif pour nous permettre d'accéder à sa configuration.

Normalement l'installation doit avoir détecté la langue de votre PC et l'IDE démarre donc en français. Si ce n'est pas le cas, il suffit de sélectionner le menu "File" puis "Preference" (version anglaise). Sur la ligne "Editor language" il nous faut sélectionner soit "System Default" pour que l'IDE utilise la langue de votre OS (Windows® - normalement cette option est activée par défaut) soit "Français (French)" si l'option précédente de fonctionne pas. Une fois le choix effectué, il faut fermer l'IDE et le relancer pour que la modification soit prise en compte.

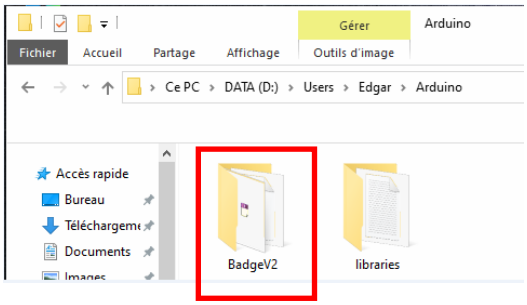
Suite à l'installation de l'IDE, un répertoire supplémentaire a été ajouté dans notre répertoire ...*{Users\...}***Arduino**. C'est dans ce répertoire "**Arduino**" que nous stockerons nos divers projets de développement. Il est important de noter que ces projets devront être installés dans un répertoire dédié portant le même nom que le fichier utilisant l'extension ".ino" spécifique à l'IDE.

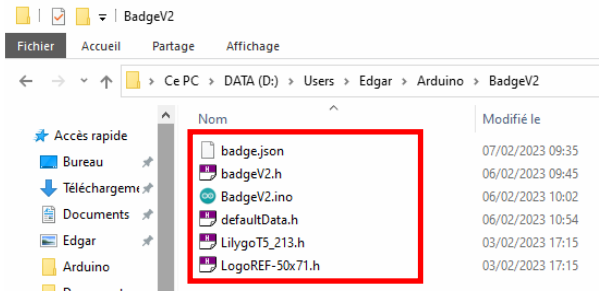
Par exemple, le fichier principal du projet du badge **E. Paper** est nommé "**BadgeV2.ino**" et il doit être situé dans le répertoire du même non qui devra être, lui, dans "{Users\...\}**Arduino**\... " comme décrit dans l'exemple ci-dessous :

*{D:\Users\Edgar}***Arduino**



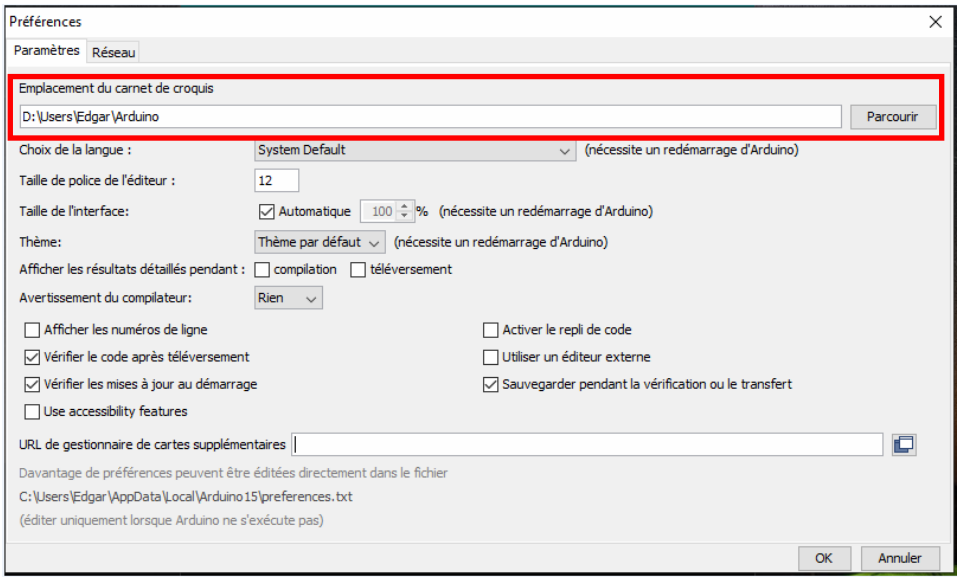
Dans ce répertoire on trouve le sous-répertoire du projet qui contient tous les fichiers nécessaires à la compilation.





En fait, ce répertoire de travail n'est pas forcément figé, il peut être modifié comme nous le souhaitons et nous pouvons le redéfinir au sein de notre IDE. Pour ce faire revenons au menu "**Préférences**" pour identifier et/ou compléter notre configuration :

"Emplacement du carnet de croquis" : Indique le répertoire de travail où seront sauvegardés nos projets de développement. Les autres options doivent être, pour le moment, conservées en l'état, c'est à dire avec les valeurs par défaut.



INSTALLATION DE LA LIBRAIRIE DE LA CARTE "ESP32 Dev Module"

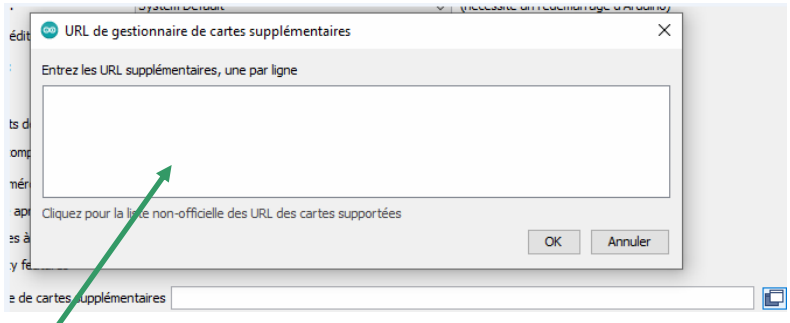
Avant de se lancer dans le téléchargement de diverses librairies complémentaires il est nécessaire de préciser certaines notions importantes. L'environnement de développement Arduino est un système ouvert et coopératif. À ce titre on a accès à une multitude de cartes électroniques (hardwares) de toutes sortes et aussi à une très grosse quantité de logiciels pour tous ces divers "hardwares". On trouve aussi des exemples, des "tutos" sur le Net, bref, il y a matière à trouver la réalisation qui nous convienne pour notre plus grand plaisir.

Pour obtenir un tel résultat, l'IDE nécessite qu'ont lui ajoute des librairies supplémentaires. Ce sont en fait des logiciels, des fichiers de configuration, des fichiers de codes à compiler, des exemples de codes, etc.

Dans notre cas, la carte utilisée par notre **E. Paper** n'est pas prise en compte par défaut lors de l'installation de l'IDE. Il nous faut donc la lui ajouter en téléchargeant toutes les librairies qui lui sont spécifiques. C'est ce que l'on va faire dans ce chapitre.

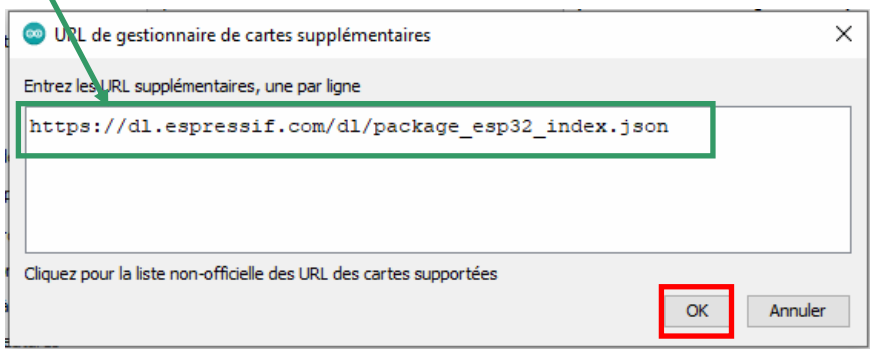
Pour récupérer les bibliothèques de notre carte "**ESP32 Dev Module**" nous allons procéder en deux étapes. La première consiste à faire reconnaître le site du fournisseur de la librairie (lien Internet) pour que le gestionnaire de carte puisse y récupérer les fichiers nécessaires. La deuxième étape sera donc le téléchargement de ces fichiers et leur installation.

Nous devons utiliser à nouveau le menu "**Préférences**" et le contrôle appelée "**URL de gestionnaire de cartes supplémentaires**". En bout de cette zone, on trouve un bouton qui nous ouvre une seconde fenêtre nous permettant de saisir du texte sur une ou plusieurs lignes.



https://dl.espressif.com/dl/package_esp32_index.json

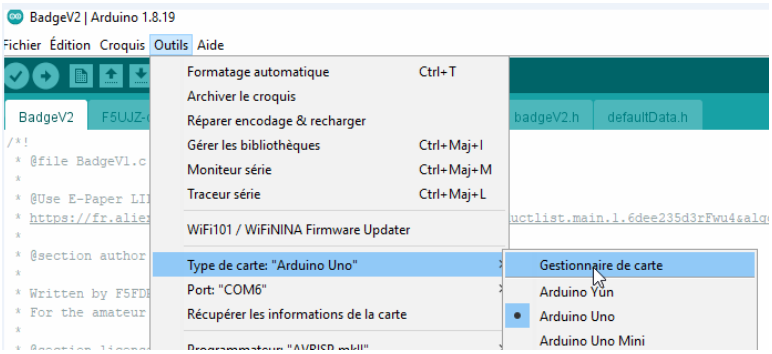
Nous allons recopier la ligne ci-dessus (copier / coller) dans la fenêtre de gestion des "**URL...**" puis valider par le bouton "**OK**".



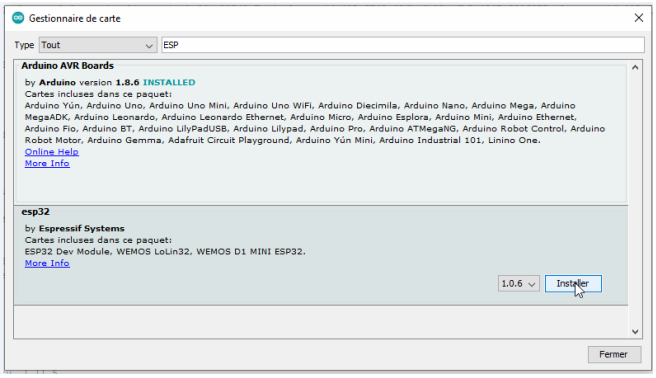
La ligne ajoutée doit maintenant figurer dans le contrôle "**URL de gestionnaire de cartes supplémentaires**". Fermer ensuite le menu "**Préférences**" par son bouton "**OK**".

Nous pouvons installer maintenant les diverses bibliothèques associées aux cartes basées sur le microcontrôleur "**ESP32**" fournies par le fabricant "Expressif".

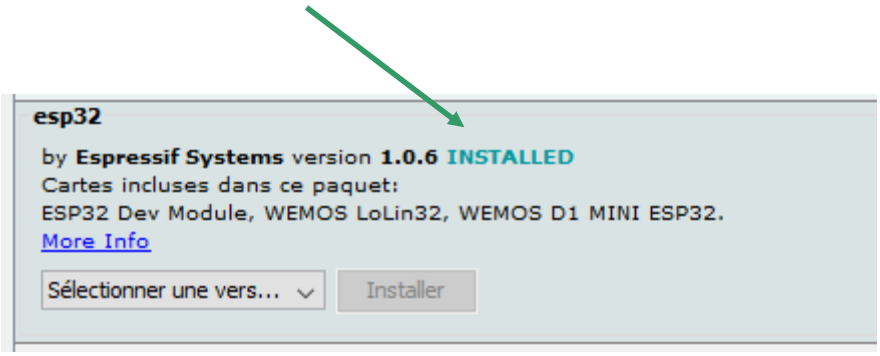
Comme pour ajouter une librairie logicielle supplémentaire, nous allons ajouter une carte (un ensemble de cartes en fait) dans notre IDE en utilisant le menu "**Outils**" puis "**Type de carte: "Arduino Uno**" et enfin "**Gestionnaire de carte**".



Dans la fenêtre du gestionnaire de carte qui vient de s'ouvrir, il nous faut sélectionner le "**Type: Tout**" et dans la zone du filtre de tri, il suffit de saisir les trois premières lettres du type de carte que l'on recherche : "**ESP**" par exemple.



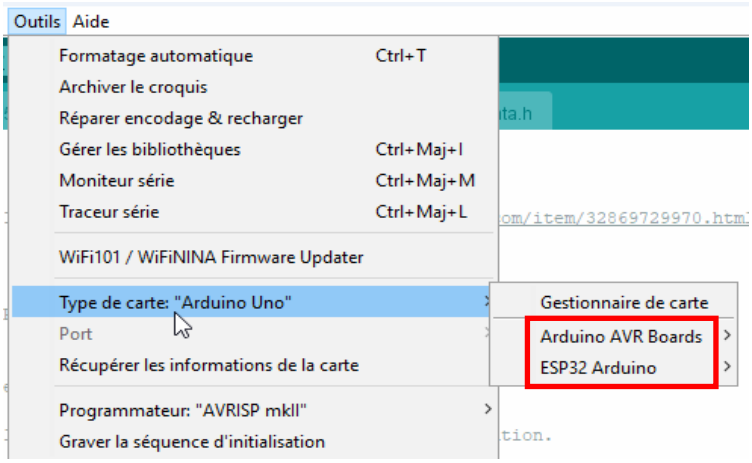
La recherche doit nous afficher deux téléchargements. Le premier est déjà installé, le second correspond au lien que nous avons ajouté au paragraphe précédent et nous permet maintenant d'installer la version "**1.0.6**" fournie par "Expressif". Sélectionnons le bouton "Installer" ; la confirmation de l'installation sera indiquée par l'affichage "**INSTALLED**" à droite de l'indice de version :



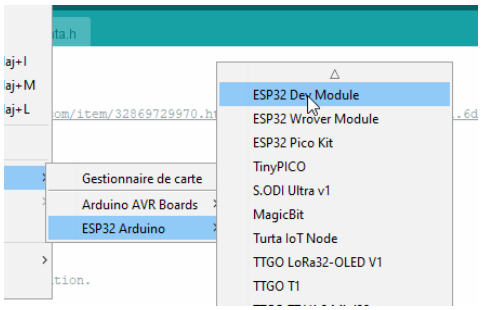
CONFIGURATION DE LA CARTE "ESP32 Dev MODULE"

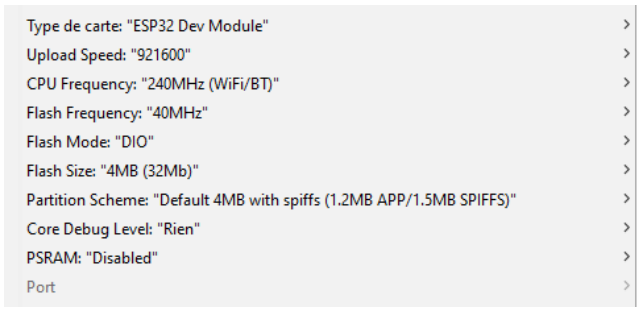
Maintenant que notre carte "**ESP32**" est prise en compte dans notre environnement de travail, nous devons configurer ses paramètres afin qu'ils correspondent exactement à notre afficheur **E. Paper**.

Il nous faut sélectionner le menu "**Outils**" puis "**Type de carte : *{carte active}***". Ce contrôle nous indique la carte actuellement active ainsi que les cartes "**Arduino AVR Boards**" installées par défaut. Notre nouvelle liste de cartes "**ESP32 Arduino**" doit aussi apparaître dans la liste.



Il nous faut sélectionner le menu "**ESP32 Arduino**" et dans la liste affichée nous devons valider la carte "**ESP32 Dev Module**" en cliquant sur la ligne correspondante.





Toutes les fenêtres doivent se refermer, ce qui nous oblige à réouvrir le menu "**Outils**". Normalement le "**Type de carte :**" doit bien indiquer notre choix, "**ESP32 Dev Module**". Il ne nous reste plus qu'à modifier deux des paramètres situés sous le type de carte :

- **Flash Frequency** : doit être à 40 MHz (par défaut 80 MHz).
- **Flash Mode**: doit être sur "DIO" (par défaut QIO).

C'est tout, il ne faut pas modifier les autres paramètres. Vous pouvez faire un test afin de voir si toutes nos modifications ont bien été prises en compte en fermant l'IDE. Rouvrez-le ensuite à nouveau avec le menu "**Outils**", la carte active doit être toujours notre "**ESP32...**".

INSTALLATION DES LIBRAIRIES ADDITIONNELLES

Maintenant, nous allons ajouter les librairies spécifiques à notre application. Elles sont obligatoires pour que notre afficheur **E. Paper** puisse fonctionner correctement et elles sont au nombre de cinq :

- GxEPD2 1.5.0 avec ses dépendances :
  - Adafruit_GFX_Library 1.11.5
  - Adafruit bus IO 1.14.1
- SD 1.2.4
- ArduinoJSON 6.20.1

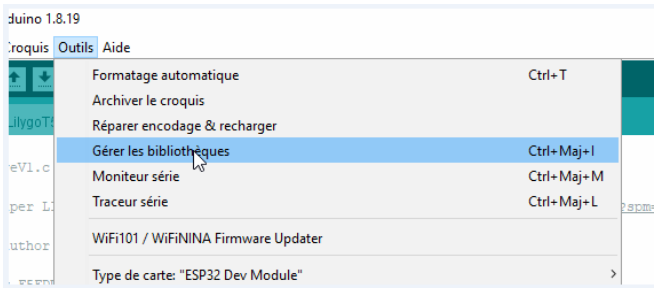
Le détail de ces librairies est le suivant :

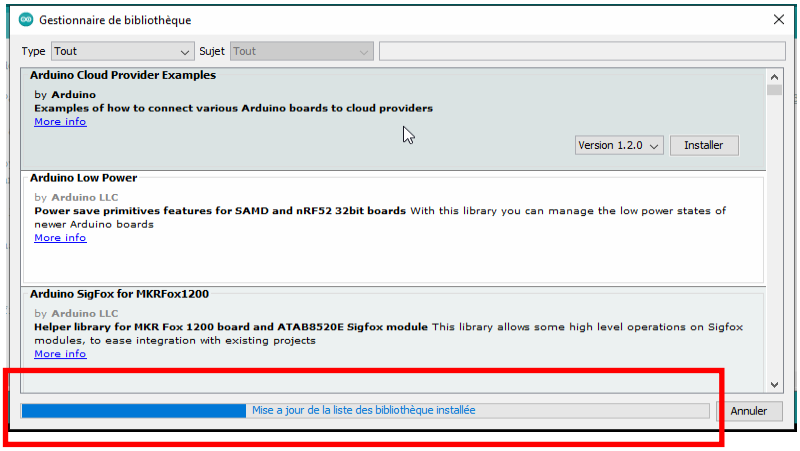
- GxEPD2 : Module de gestion des afficheurs de type E. Paper (multitude de modèles différents).
- Adafruit GFX Library : Module de gestion en mode graphique des afficheurs.
- Adafruit bus IO : Module de gestion des entrées/sorties.
- SD : Module de gestion du support de carte SD.
- ArduinoJson : Module d'interprétation et de gestion des fichiers au format JSON (utilisé pour lire la configuration de notre badge EPaper).

Comme pour le téléchargement des fichiers de notre carte "**ESP32"**, l'IDE contient un gestionnaire de librairies que nous devons utiliser pour récupérer les fichiers manquants.

Il faut retenir aussi que lors du chargement de certaines librairies, une analyse des dépendances est faite pour ajouter certains fichiers supplémentaires. Les dépendances correspondent en fait à la nécessité d'ajouter à une librairie principale une ou plusieurs autres librairies obligatoires pour que notre librairie principale puisse fonctionner. La gestion est faite automatiquement par l'IDE et lorsque la question nous sera posée, il nous faudra toujours répondre "**Oui**" pour autoriser les téléchargements supplémentaires. Dans le cas contraire, on court le risque d'avoir des erreurs de compilation à cause de bibliothèques introuvables ou manquantes.

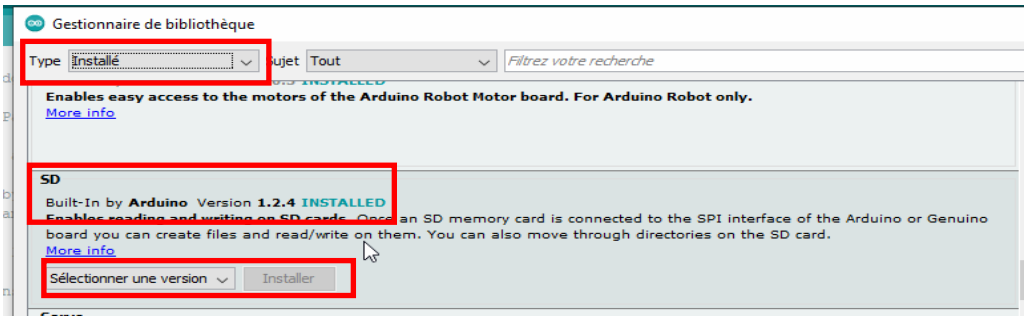
Le menu "**Outils**" puis "**Gérer les bibliothèques**" (ou le raccourci CTRL+MAJ+I) va nous faire afficher l'utilitaire nous permettant de gérer l'ensemble des librairies déjà installées et/ou à installer. Il est à noter qu'il nous faut patienter quelques dizaines de secondes pour que cet utilitaire s'exécute et qu'il puisse parcourir la liste des fichiers déjà présent dans l'IDE.



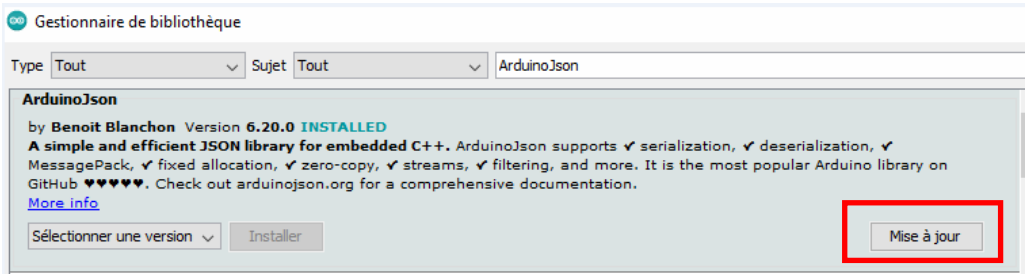


Une fois l'outil correctement initialisé, nous pouvons y accéder en vérifiant, dans un premier temps, les librairies installées. Pour ce faire nous devons utiliser le "ComboBox: **Type**" et sélectionner le choix "**Installé**".

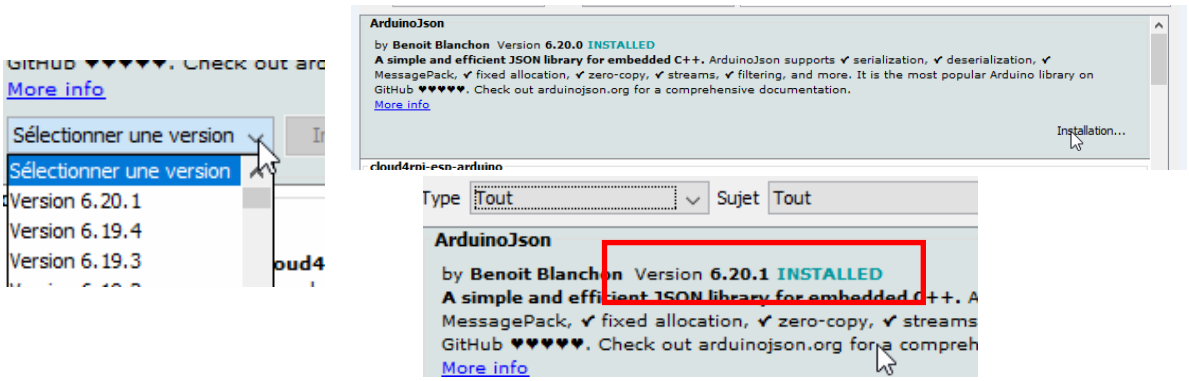
A l'aide de la souris du PC nous pouvons faire défiler cette liste et vérifier les bibliothèques déjà présentes. Par exemple, la gestion de la carte SD est bien présente dans notre IDE et elle est bien à la dernière version requise pour notre application. En cas de besoin, mais c'est rarement le cas, il nous est possible de revenir sur une version antérieure en utilisant le "ComboBox: **Sélectionner une version**". Il est donc important de bien autoriser le "Firewall" comme expliqué lors de la configuration de notre IDE. Dernière chose : le bouton "**Installer**" doit généralement être grisé et non utilisable. Il ne sert que pour permettre l'installation d'une version antérieure ou postérieure à celle active.



Ajoutons maintenant les librairies manquantes en sélectionnant le "ComboBox: **Tout**" et dans la barre "***Filtrez votre recherche***" il nous suffit de mettre le nom de la librairie à trouver. L'exemple ci-dessous décrit cette mécanique de recherche qui prend aussi en compte la mise à jour éventuelle des fichiers. Il concerne la bibliothèque "**ArduinoJson**" qui, lors de la création du projet, était à la version "**6.20.0**". Entretemps, la version a été mise à jour et l'IDE nous propose de la télécharger.



On a donc le bouton "**Mise à jour**" qui apparaît si on positionne le curseur de la souris sur le cadre de cette librairie. Si on utilise le "ComboBox: **Sélectionner une version**", on constate qu'il y a une nouvelle version "**6.20.1**" la version courante étant la "**6.20.0**" qui n'est donc pas dans cette liste puisqu'elle est active. Conservons ce "ComboBox" sur la position initiale et validons cette mise à jour par le bouton "**Mise à jour**".

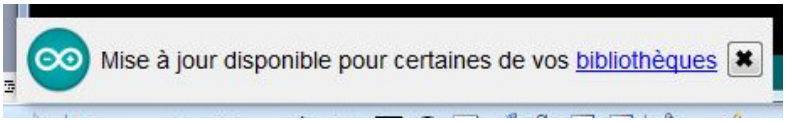


Procéder à l'identique pour les autres librairies nécessaires et décrites dans ce chapitre. Bien entendu, si certaines doivent être mises à jour, n'hésitez pas, car entre la date de création de cette notice et son utilisation, il est fort probable, comme pour la bibliothèque "**ArduinoJson**", qu'il y ait de nouvelles versions de fichiers à télécharger.

INFORMATIONS SUR LES MISES A JOUR AUTOMATIQUES

Nous avons vu dans les chapitres précédents la nécessité d'autoriser le "Firewall" pour qu'il laisse passer les requêtes de connexions à Internet demandées par notre IDE. Aujourd'hui, sans cette possibilité, il nous serait impossible de mener à bien un projet "**Arduino**". Cela dit, il faut retenir qu'à chaque nouveau démarrage de l'IDE, un processus (caché) est exécuté afin d'identifier les mises à jour disponibles pour les diverses librairies installées et aussi pour les cartes (hardware) prises en compte par l'environnement de développement. Ce processus peut être aussi déclenché lorsqu'on utilise le menu "**Préférences**".

C'est pour cette raison que pendant l'utilisation de l'IDE, certains messages "**pop-up**" peuvent apparaître pendant quelques secondes. Ils nous indiquent que certaines bibliothèques peuvent être mises à jour ainsi que certaines "cartes" présentes dans notre IDE. Il est important de retenir que ce sont des messages d'information et qu'en aucun cas ces mises à jour seront effectuées automatiquement. En fait, ces messages proposent un lien "hypertexte" qui nous permettra d'exécuter la (les) mises à jour recommandées. Ces messages resteront affichés une petite dizaine de secondes.



Les raisons de ces mises à jour sont les suivantes:

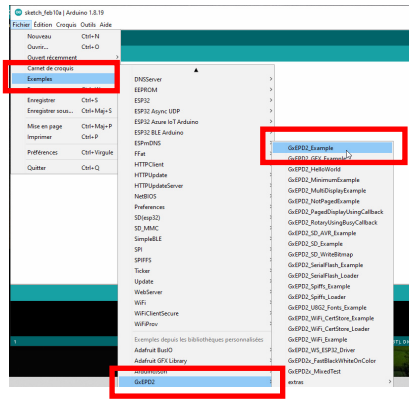
- Correction de "bugs" dans les librairies.
- Ajout de nouvelles fonctionnalités pour les librairies existantes.
- Correction ou ajout de "pilotes" pour les cartes installées
- Optimisation de certains codes (vitesse d'exécution, empreinte mémoire)...
- ...

COMPILATION D'UN PROGRAMME DE TEST

Pour finir avec cette notice, nous allons vérifier que notre environnement de développement est bien installé et opérationnel afin de nous permettre de programmer notre **E. Paper**.

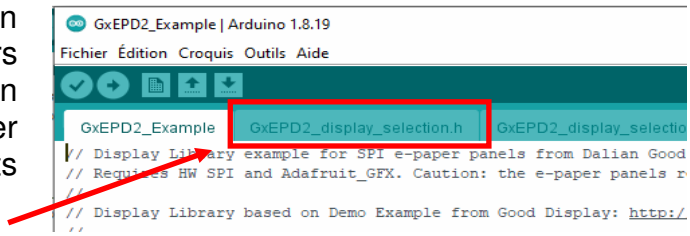
Nous allons charger un exemple de programme et le compiler sans l'injecter dans notre cible. Le but est ici de vérifier qu'il ne manque pas de librairie pour la compilation de notre projet "**BadgeV2**". De plus, l'ouverture d'un fichier d'exemple va aussi nous permettre de "mesurer" un petit peu l'étendue des possibilités de notre "**IDE Arduino**".

Depuis notre "sketch" par défaut ouvert au début de cet article nous allons utiliser le menu "**Fichier**" puis "**Exemples**". Dans la liste des exemples disponibles et affichés, il nous faut descendre vers le bas de cette liste et sélectionner "**GxEPD2**" (une des librairies que nous avons ajoutée au chapitre précédent). Une fois ce menu sélectionné, une liste d'exemples nous est proposée et nous allons choisir le premier, c'est à dire "**GxEPD2_Example**".



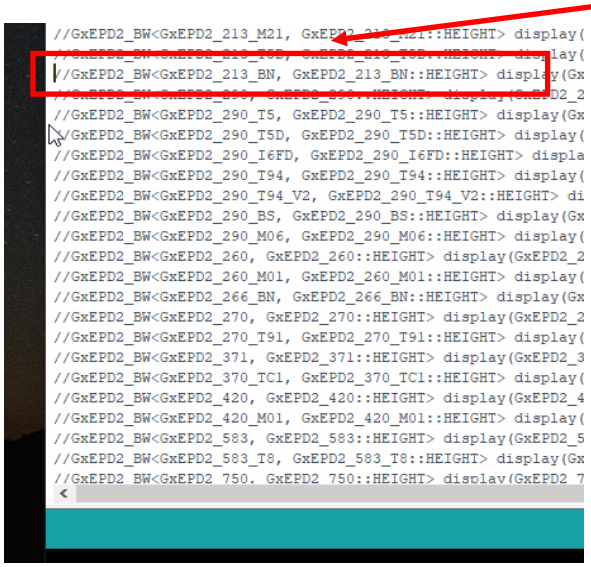
Le simple fait de sélectionner cet exemple nous ouvre une seconde instance de l'IDE avec tous les fichiers du projet associés à cet exemple. Nous pouvons donc fermer maintenant notre premier "sketch" par défaut et ne conserver que le nouveau. On a pu constater que le nombre d'exemples est quand même conséquent et qu'il y a bien matière à "bidouiller" ou à explorer.

Si on lançait la compilation maintenant nous aurions des erreurs car nous n'avons pas défini un "afficheur" type. Il faut donc modifier une ligne dans un des fichiers ouverts par notre "sketch".

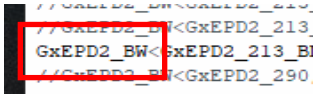


Sélectionnons l'onglet "**GxEPD2_display_selection.h**".

Ensuite, à l'aide de la souris ou du clavier, descendons jusqu'à la ligne **168** ; le compteur de ligne est situé en bas de la fenêtre et à gauche.



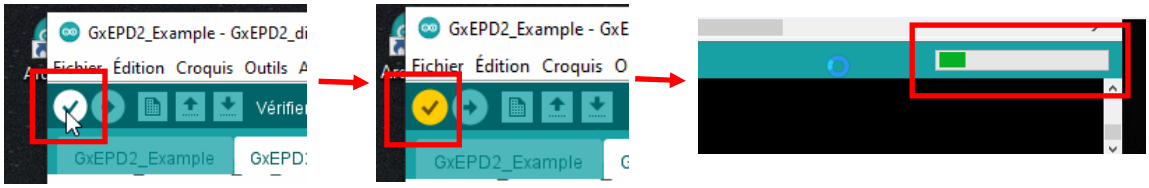
Et une fois cette ligne atteinte, il nous faut supprimer les deux "//" en début de la ligne devant le texte "GxEPD2...". Nous utiliserons la touche Suppression afin que la ligne soit bien "calée" à gauche.



Il est inutile de sauvegarder la modification pour deux raisons :

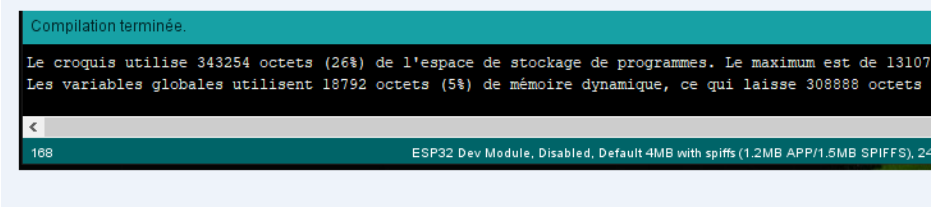
- Les fichiers de l'exemple sont "tagés" en lecture seule, il nous sera impossible de sauvegarder la modification dans le répertoire d'origine des exemples.
- Ce n'est qu'un test de fonctionnement et il est donc inutile de sauvegarder cet exemple dans notre répertoire de travail.

Nous pouvons maintenant lancer la compilation de notre exemple en sélectionnant le bouton "**Vérifier**". La compilation va prendre un peu de temps, plusieurs minutes selon le PC utilisé. Le bouton "**Vérifier**" doit changer de couleur et la barre de progression, en bas à droite de la fenêtre, doit nous indiquer le bon déroulement de la compilation.



La compilation est terminée lorsque le texte suivant est affiché dans la fenêtre de dialogue :

Le croquis utilise 343254 octets (26%) de l'espace de stockage de programmes. Le maximum est de 1310720 octets.
Les variables globales utilisent 18792 octets (5%) de mémoire dynamique, ce qui laisse 308888 octets pour les variables locales. Le maximum est de 327680 octets.



Dans le cas contraire, si nous obtenons un message de couleur orange, c'est un rapport d'erreur. Cela nous oblige à reprendre les étapes de l'installation afin d'identifier celle qui n'a pas été effectuée convenablement. Il est nécessaire de lire et de comprendre les messages d'erreur, et de corriger chaque point correspondant.

CONCLUSION

Notre IDE est fin prêt à être utilisé avec notre afficheur **E. Paper**, pour notre plus grand plaisir. De plus, nous avons acquis une nouvelle compétence qui nous permet d'envisager de reproduire cette démarche pour un autre "gadget Arduino" que l'on peut trouver dans une revue d'électronique quelconque ou via Internet. Le but est de permettre la compilation d'un programme pour une interface (carte) que l'on souhaite utiliser sans être obligé de créer du "code programme" mais utiliser le code existant.

Il existe une multitude d'applications : thermomètre digital, détecteur de présence, télécommande, fréquencemètre, lecteur RFID, etc. La liste est trop importante pour être décrite ici. Le principe est toujours le même : on choisit une "carte support", on y ajoute d'éventuels accessoires (capteurs, boutons, etc.) et on récupère le "code programme" qu'une personne a mis gracieusement à notre disposition (coopération). On trouve sur le Net une multitude d'exemples, il n'y a qu'à se servir...

Il existe d'autres méthodes pour injecter du "code programme" dans une cible mais ces méthodes ne permettent pas de modifier légèrement une partie de ce code afin de l'adapter à nos besoins. Généralement, ils utilisent un bloc de données binaires déjà compilé et donc non modifiable. Pour le cas de notre **E. Paper**, la solution d'utiliser l'IDE nous permet de programmer notre carte et nous donne, de ce fait, la possibilité de définir nos propres textes à afficher sans avoir besoin d'une carte SD.

Bonne réalisation à tous, en espérant que cette notice vous aura apporté une nouvelle façon d'appréhender les réalisations à base de micro-contrôleurs.

Edgar - F5FDR
2023