15/07/2022

Développer sur le nœud SoLo

De la récupération du logiciel sur le git aux premiers développements

Maxime Rubéo-Lisa
CONNECSENS

Table des matières

Table des figures 2

I. Introduction 4

II. Les outils 5

2.1. Gestionnaire de version 5

2.2. L'IDE Atollic TRUEStudio 6

2.3. Les outils de débug 7

III. Le git lorawan-node-stepat 9

3.1. Introduction 9

3.2. L'architecture 10

3.3. Récupérer le git 12

IV. Développer sur le nœud SoLo 16

4.1. Prendre une bonne habitude avec GIT 16

4.2. Modification sur le software 16

4.3. Commencer à travailler avec l'IDE Atollic TrueStudio 17

4.4. Utiliser les outils de débug 21

4.3.1. Convertisseur USB-TTL 22

4.3.2. Programmeur ST-Link/V2-Isol 23

V. Ajouter un nouveau driver 29

5.1. Création des fichiers du driver 29

5.2. Fonctions nécessaires au fonctionnement du driver 34

Table des figures

Figure 1 : SourceTree 5
Figure 2 : IDE TrueSTUDIO 6
Figure 3 : Barre d'outil IDE TrueSTUDIO 6
Figure 4 : USB vers TTL (debbuger) 7
Figure 5 : Programmeur ST-LINK/V2-ISOL 7
Figure 6 : Logs du nœud sur une invite de commande Windows 8
Figure 7 : Page d'accueil forge UCA 9
Figure 8 : branche master du git "Lorawan-node-stepat" 10
Figure 9 : architecture git 11
Figure 10 : Récupérer le dépôt git SourceTree 1 12
Figure 11 : RECUPERER LE DEPOT GIT SOURCETREE 2 13
Figure 12 : RECUPERER LE DEPOT GIT SOURCETREE 3 14
Figure 13 : Dossier git créé 14
Figure 14 : Numéro de version du software connecsens 16
Figure 15 : fichier changelog.txt 17
Figure 16 : Chemin de l'espace de travail de l'IDE 18
Figure 17 : Sélectionner son projet 1 18
Figure 18 : Sélectionner son projet 2 19
Figure 19 : Emplacement "restore" pour le volet de navigation dans l'architecture du projet 19
Figure 20 : Page d'accueil de l'IDE TrueSTUDIO 20
Figure 21 : Builder courant 20
Figure 22 : Fichier binaire 21
Figure 23 : Connectique USB-TTL 22
Figure 24 : Configuration lecture des logs 22
Figure 25 : Commande pour lancer le script de lecture des logs sur une invite de commande Windows 23
Figure 26 : Logs du nœud sur une invite de commande Windows 23
Figure 27 : Port JTAG du nœud 24
Figure 28 : Brochage du connecteur JTAG 24
Figure 29 : Connectique entre la borne JTAG du nœud et le programmeur 25
Figure 30 : Debug configuration 26
Figure 31 : Run external tools 27
Figure 32 : Configuration du run external tools 28
Figure 33 : Architecture Sensors 29
Figure 34 : Emplacement fichiers driver 30
Figure 35 : Modification nom fichier hpp 31
Figure 36 : Nom après modification 32
Figure 37 : Nom fichier CPP 33
Figure 38 : Ajout header au fichier factory.hpp 33
Figure 39 : ajout instance fichier factory.hpp 34
Figure 40 : Ajout directive préprocesseur fichier config.h 34
Figure 41 : Contenu header du driver 35

Révision	Date	Modifications
1.0	01/07/2022	Première version (MRL)
1.1	15/07/2022	Ajout de quelques précisions suivant les commentaires de Laure Moiroux

I. Introduction

Ce document a pour but d'expliquer comment commencer un développement sur le nœud SoLo et des bonnes pratiques à avoir lors de ses développements.

Nous aborderons notamment les outils utiles et/ou nécessaire pour développer sur le nœud. Nous parlerons également du répertoire git qui contient tout le projet du nœud SoLo ainsi que de la démarche à suivre pour développer sur le nœud. Il faut noter que ce document n'a pas vocation à vous expliquer comment est conçu le programme du nœud, mais à expliquer la démarche à suivre avant et après votre développement. En support complémentaire à ce document je recommande de lire les 2 tutoriel sur le développement sur les nœuds fait par Jérôme Fuchet, disponible sur le drive (ConnecSenS / _instruments / _noeud_SoLo / formations_internes).

II. Les outils

Cette partie a pour objectif de vous présenter brièvement les outils utilisés pour développer sur le nœud.

2.1. Gestionnaire de version

Pour récupérer le dépôt git, vous aurez besoin d'un logiciel de gestion de version. Sous linux, une simple ligne de commande permet de récupérer un dépôt, si vous avez git d'installé.

Sous Windows, plusieurs logiciels existent. Vous pouvez utiliser celui de votre choix. Pour ce tutoriel je vais utiliser SourceTree. La démarche restera similaire qu'importe votre logiciel de gestion de version.

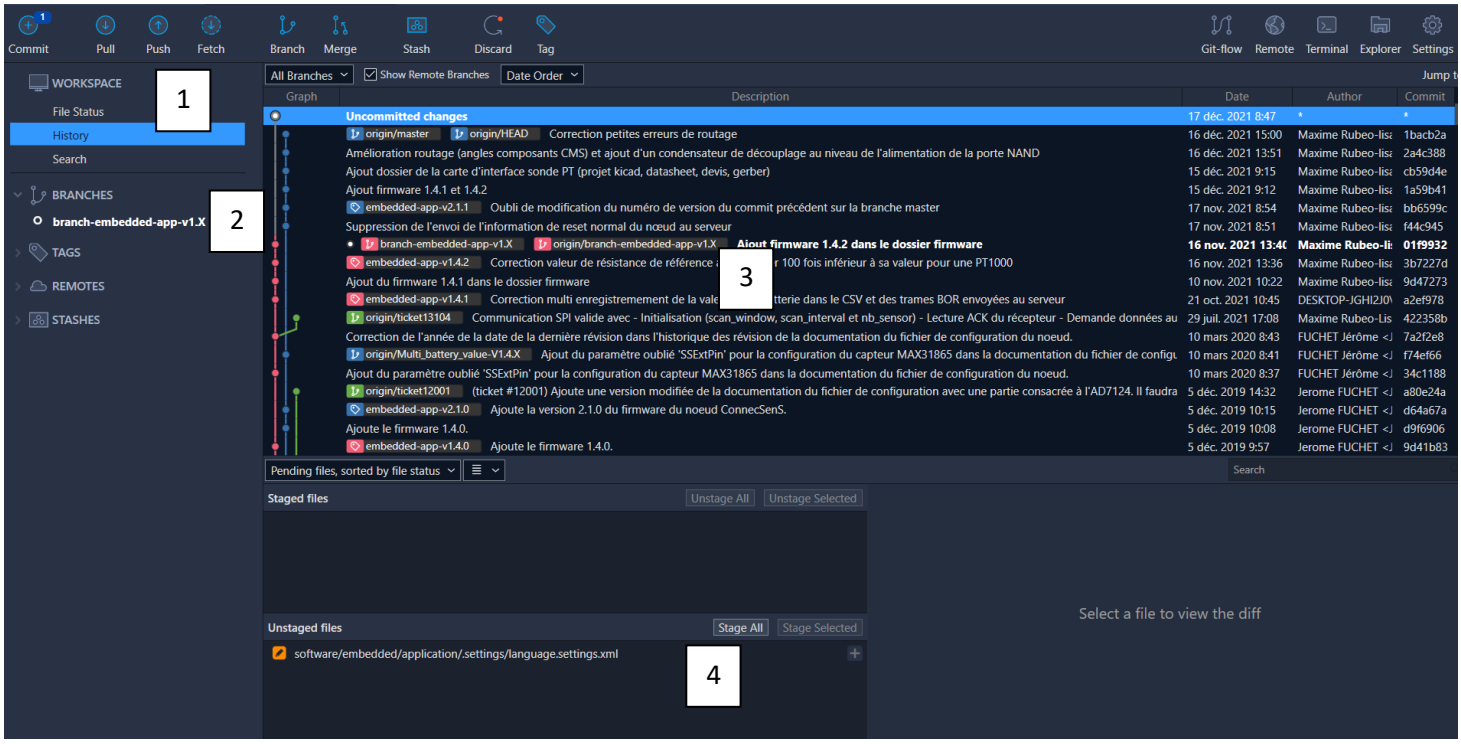

FIGURE 1 : SOURCETREE

Sur ce logiciel on retrouve les commandes permettant d'utiliser Git (1), la branche sur laquelle on travaille (2), une interface graphique (3) qui donne l'historique de toutes les modifications apportées au nœud, ainsi que ses différentes branches, et enfin les fichiers modifiés (4).

Nous verrons dans la partie 3.3 comment utiliser ce logiciel pour récupérer le git.

2.2. L'IDE Atollic TRUEStudio

Atollic est un Integrated Development Environment, soit un environnement de développement intégré. Tous les logiciels nécessaires pour développer sur le nœud sont sur le même logiciel, comme l'éditeur de texte dédié à la programmation, le compilateur, le débogueur, etc.. Il est basé sur l'IDE Eclipse et se télécharge sur le site de ST (https://www.st.com/en/development-tools/truestudio). Le projet utilise la version 9.3.

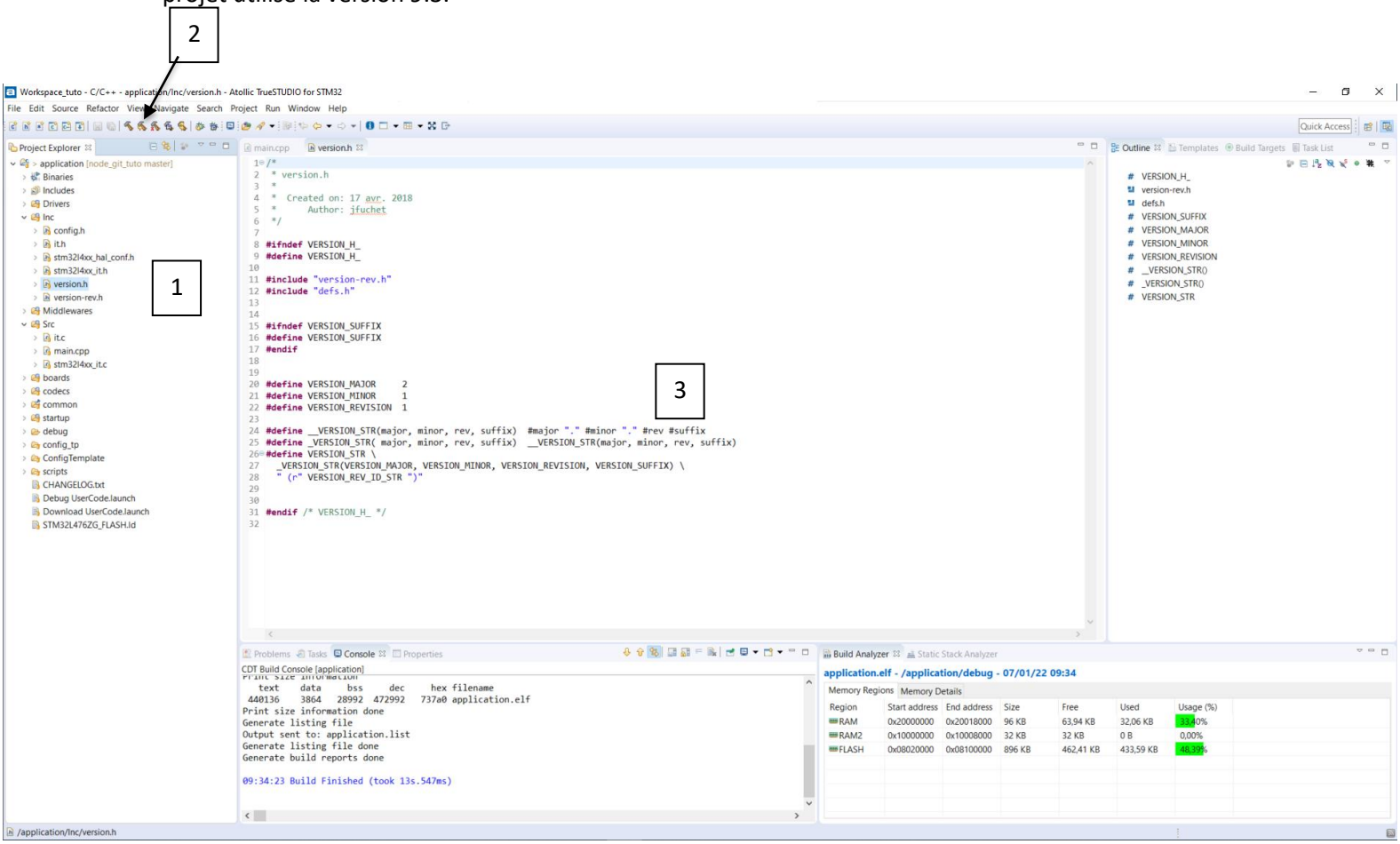

FIGURE 2 : IDE TRUESTUDIO

L'architecture du projet est dans la fenêtre de gauche (1). On retrouve l'icône pour compiler (2) et l'éditeur de texte (3).

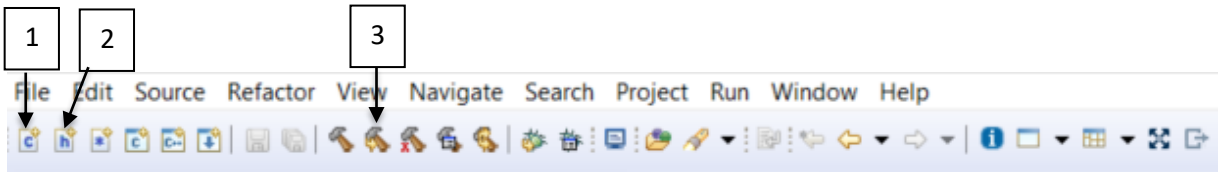

FIGURE 3 : BARRE D'OUTIL IDE TRUESTUDIO

On retrouve sur la barre d'outils 3 icône importantes :

- Ajout d'un fichier source C ou C++ (1)
- Ajout d'un fichier d'en-tête (2)

- Recompilation du projet courant (3)

2.3. Les outils de débug

Il existe 2 outils pour le débug du nœud :

- Un programmeur (ST-Link/V2-Isol)
- Logique USB vers TTL (câble USB bleu, en stock à l'INRAE)

Ces deux outils sont complémentaires mais peuvent être utilisés indépendamment.

FIGURE 4 : USB VERS TTL (DEBBUGER)

FIGURE 5 : PROGRAMMEUR ST-LINK/V2-ISOL

Le convertisseur USB vers TTL permet de lire sur une invite de commande les logs du nœud :

```
2022-02-14 13:54:00|DEBUG|connecsens|1077|Internal power: ENABLE
2022-02-14 13:54:00|DEBUG|connecsens|1112|External interruptions power: DISABLE
2022-02-14 13:54:00|DEBUG|connecsens|302|Opening datalog file 'env/datalogs/434E535301E31302-sensors.cnssrf.dlog'...
2022-02-14 13:54:00|DEBUG|datalog_cnssrf|227|Datalog indexes file is: env/datalogs/434E535301E31302-sensors.cnssrf.dlog.
indexes
2022-02-14 13:54:00|DEBUG|connecsens|305|CNSSRF datalog file 'env/datalogs/434E535301E31302-sensors.cnssrf.dlog' has bee
n opened.
2022-02-14 13:54:00|DEBUG|connecsens|938|Node reseted because powered on or BOR.
2022-02-14 13:54:00|INFO |connecsens|444|CNSSRF payload: C18400781B9D290F3811242BD07BD82B42842093F901
2022-02-14 13:54:00|DEBUG|connecsens|452|CNSSRF meta data: 012015400561036105610761
2022-02-14 13:54:00|DEBUG|sensor|557|TempHumi: Opening...
2022-02-14 13:54:00|DEBUG|sensor|558|TempHumi: Opened.
2022-02-14 13:54:00|DEBUG|sensor|586|TempHumi: Reading...
2022-02-14 13:54:00|INFO |sensor|590|TempHumi: Reading done.
2022-02-14 13:54:00|DEBUG|sensor|577|TempHumi: Closed.
2022-02-14 13:54:00|DEBUG|sensor|557|Pressure: Opening...
2022-02-14 13:54:00|DEBUG|sensor|558|Pressure: Opened.
2022-02-14 13:54:00|DEBUG|sensor|586|Pressure: Reading...
2022-02-14 13:54:00|INFO |sensor|590|Pressure: Reading done.
2022-02-14 13:54:01|DEBUG|sensor|577|Pressure: Closed.
2022-02-14 13:54:01|DEBUG|sensor|557|Lumi: Opening...
2022-02-14 13:54:01|DEBUG|sensor|558|Lumi: Opened.
2022-02-14 13:54:01|DEBUG|sensor|586|Lumi: Reading...
2022-02-14 13:54:01|INFO |sensor|590|Lumi: Reading done.
2022-02-14 13:54:01|DEBUG|sensor|577|Lumi: Closed.
2022-02-14 13:54:01|INFO |sensor|401|Lumi: Sensor's state file does not exist; use default state.
2022-02-14 13:54:01|INFO |connecsens|444|CNSSRF payload: C18300781B9D290F3811242BD07BD80B2346A9422502930B05411223769AA85
104E8681A230FAD26D008D200
```

FIGURE 6 : LOGS DU NŒUD SUR UNE INVITE DE COMMANDE WINDOWS

III. Le git lorawan-node-stepat

3.1. Introduction

Ce répertoire git sert à conserver tout le travail réalisé sur le nœud SoLo. Il permet également de travailler à plusieurs sur le nœud, sans se gêner.

Si vous souhaitez utiliser ce git, il faut pouvoir se connecter sur la forge UCA avec un compte UCA ou un compte invité. Vous devez demander l'accès à ce projet. Pour cela vous devrez en faire la demande à M. Royer Laurent qui gère le projet ConnecSens.

Une fois que vous êtes sur la forge UCA dans le projet lorawan-node-stepat, vous vous retrouvez sur la page illustrée sur la figure suivante :

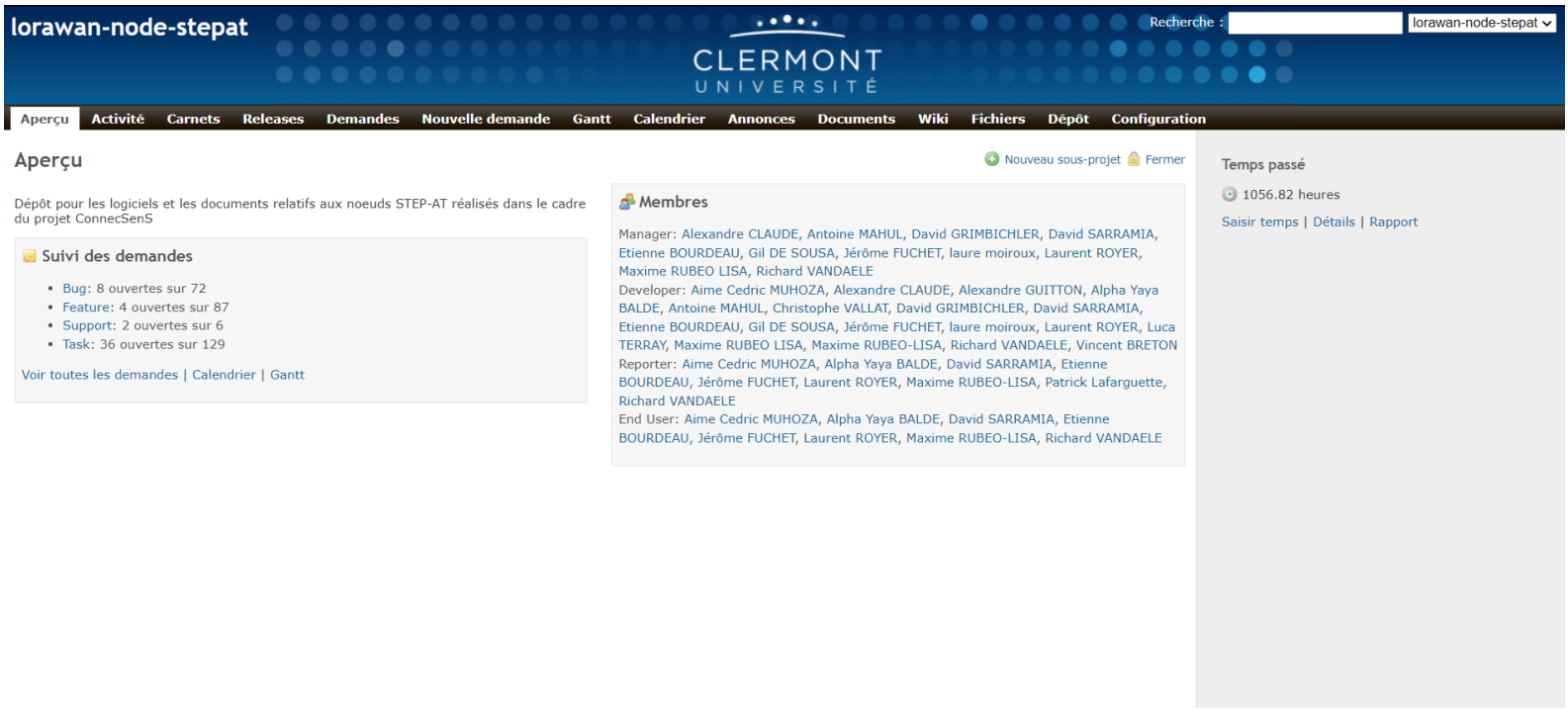

FIGURE 7 : PAGE D'ACCUEIL FORGE UCA

Sur le bandeau supérieur vous retrouvez différents onglets. Ceux qui vont être les plus importants sont :

- Aperçu
- Activités
- Carnets
- Demandes
- Nouvelle demande
- Dépôt

Aperçu donne une vision globale du projet, avec le nombre de tâches d'ouvertes ainsi que leur nature (bug, feature, support ou task) et les différents utilisateurs avec leur fonction.

Activités donne un historique de tout ce qui a été fait sur le nœud.

Carnet est une liste de tâches ouvertes.

Demandes est un historique de toutes les demandes faites.

Nouvelle demande permet de créer une nouvelle demande, en lui attribuant une nature, une personne chargée de résoudre la tâche et de donner quelques précisions.

Dépôt est le repository du projet, que l'on peut voir sur l'image suivante.

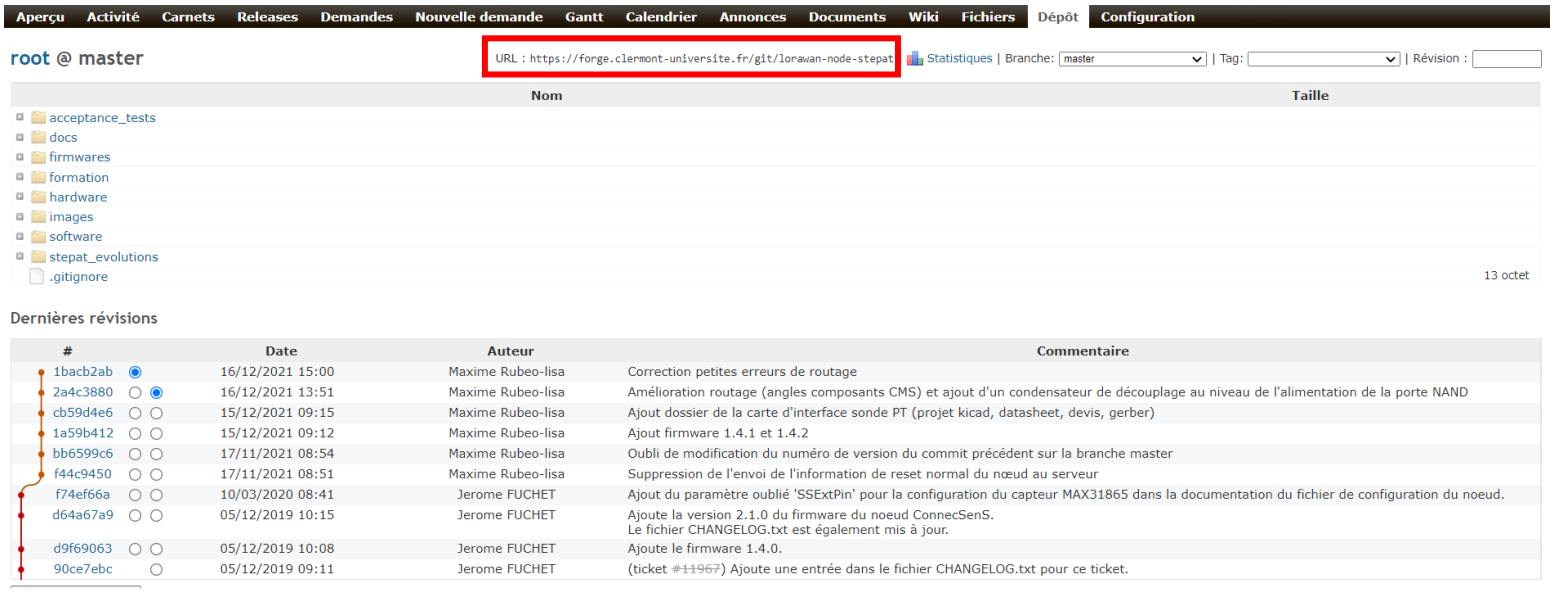

FIGURE 8 : BRANCHE MASTER DU GIT "LORAWAN-NODE-STEPAT"

Dans l'encadré on retrouve l'URL nécessaire pour récupérer tout le projet GIT.

3.2. L'architecture

On peut distinguer 2 branches principales :

- Master
- Branch-embedded-app-v1.X

Ces deux branches contiennent les 2 versions du logiciel du nœud.

La version 2, abrégée en 2.X.X est disponible sur la branche master. La version 1 est disponible sur la branche Branch-embedded-app-v1.X. La différence entre les deux versions est que le nœud en version 1 redémarre de 0 à chaque réveil. Il va relire le JSON et relancer tous les programmes. Le nœud ne se réveille pas sur un début de communication. La version 2 du programme permet de conserver l'état du nœud entre chaque réveil.

Les autres branches sont des branches de travail. Par convention lorsqu'on fait une modification on crée une branche, surtout si on effectue plusieurs tâches en parallèle.

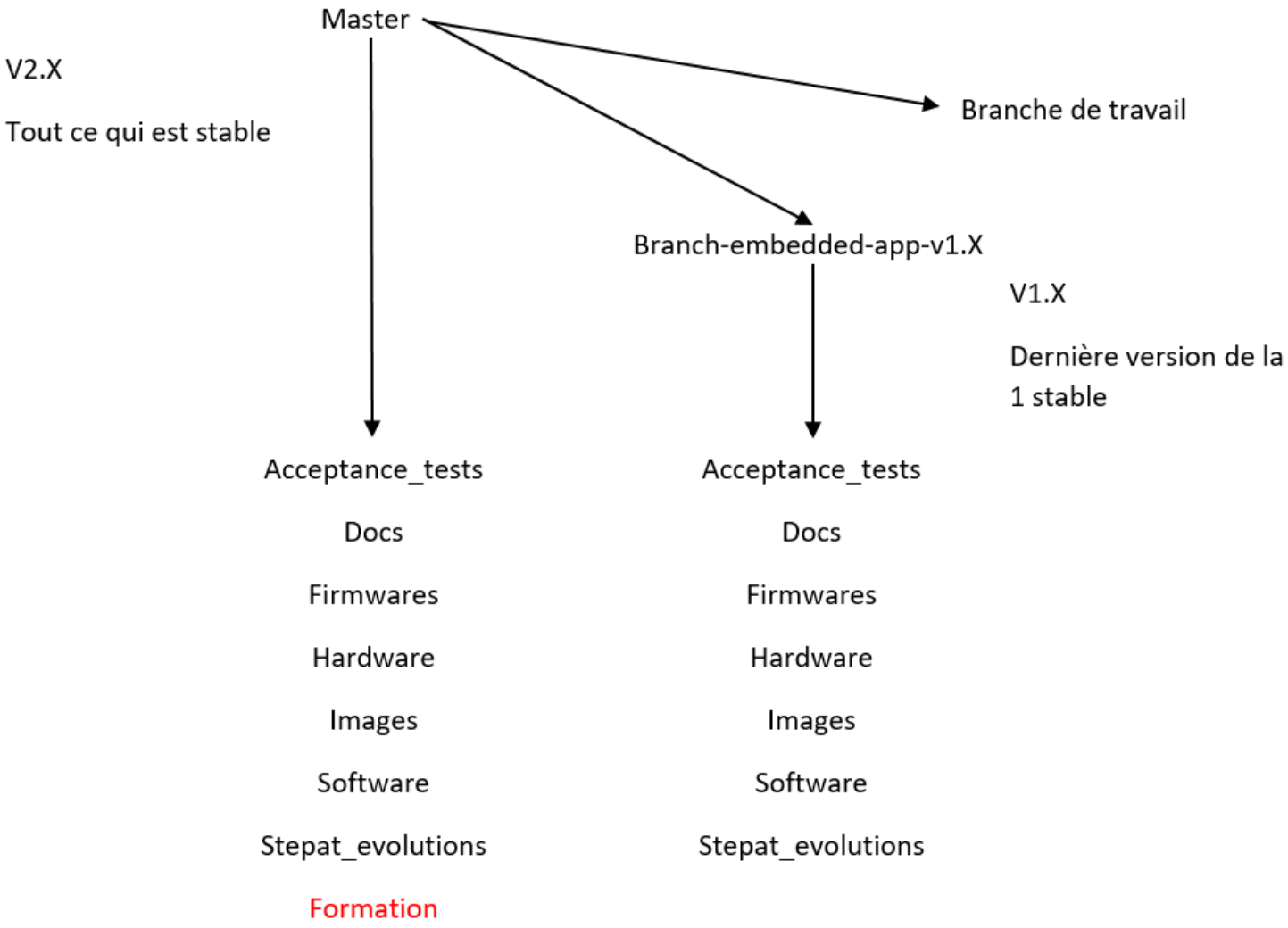

FIGURE 9 : ARCHITECTURE GIT

- Acceptance_tests : tests nœud
- Docs
- Firmwares : toutes les versions soft, dispo sur le drive
- Hardware : projets hardware du nœud
- Images : logos, etc
- Software : logiciel de développement du nœud
- Stepat_evolutions : liste des évolutions souhaitées sur 1ère et 2nd prod
- Formation : disponible uniquement sur la branche master

3.3. Récupérer le git

Pour récupérer l'archive git, ouvrez un nouvel onglet dans SourceTree, puis cliquer sur clone

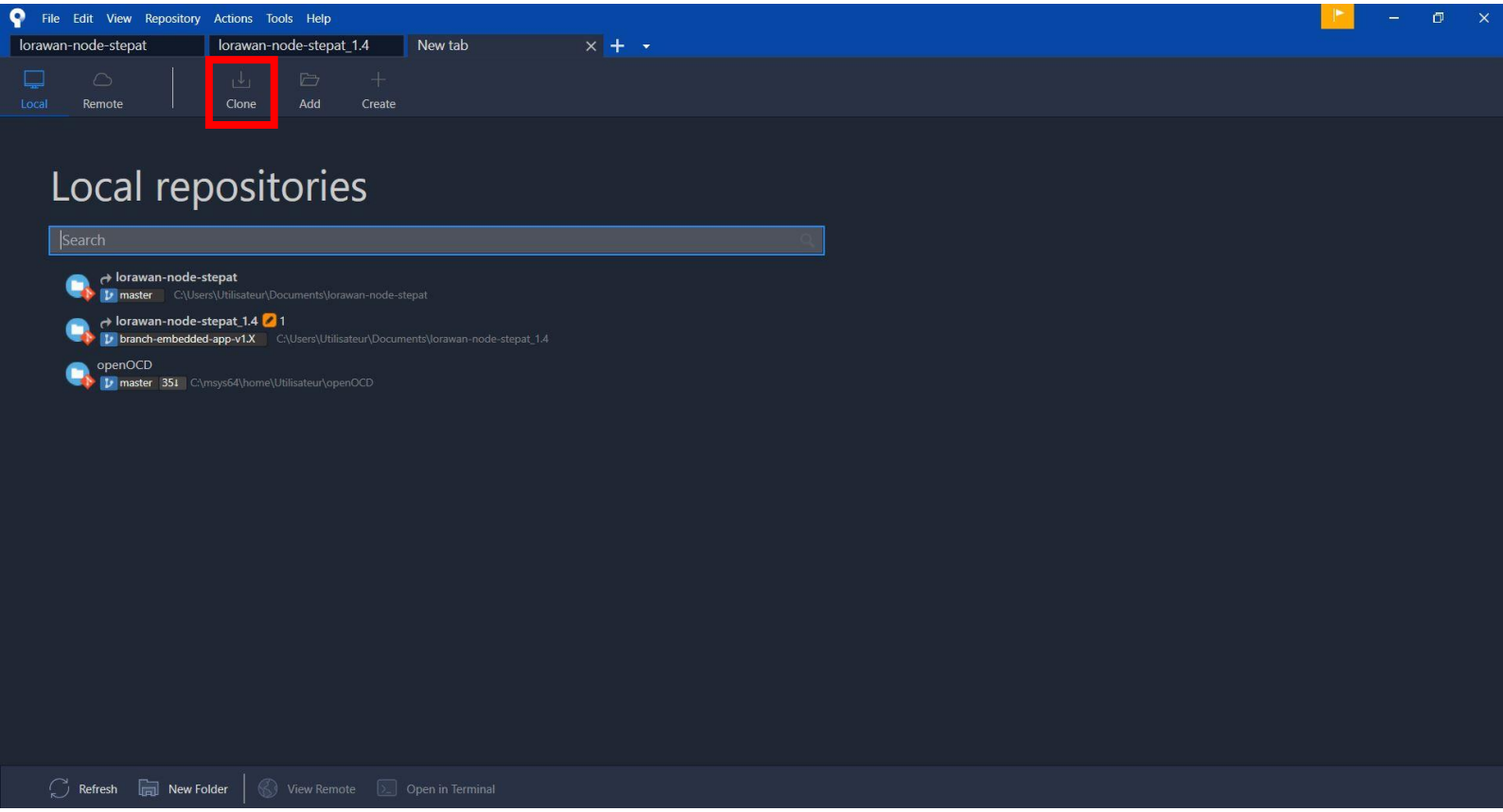

FIGURE 10 : RECUPERER LE DEPOT GIT SOURCETREE 1

Il faut ensuite entrer l'URL (1) renseigné sur la forge. Il faudra se logger avec mon compte UCA (2). On pourra également définir le chemin (3) et le nom du dossier (4). Dans cette exemple, je vais modifier le nom du dossier par « node_git_tuto », ce qui modifiera automatiquement le nom du dossier dans le chemin. On cliquera ensuite sur « clone » (5) pour récupérer le dépôt.

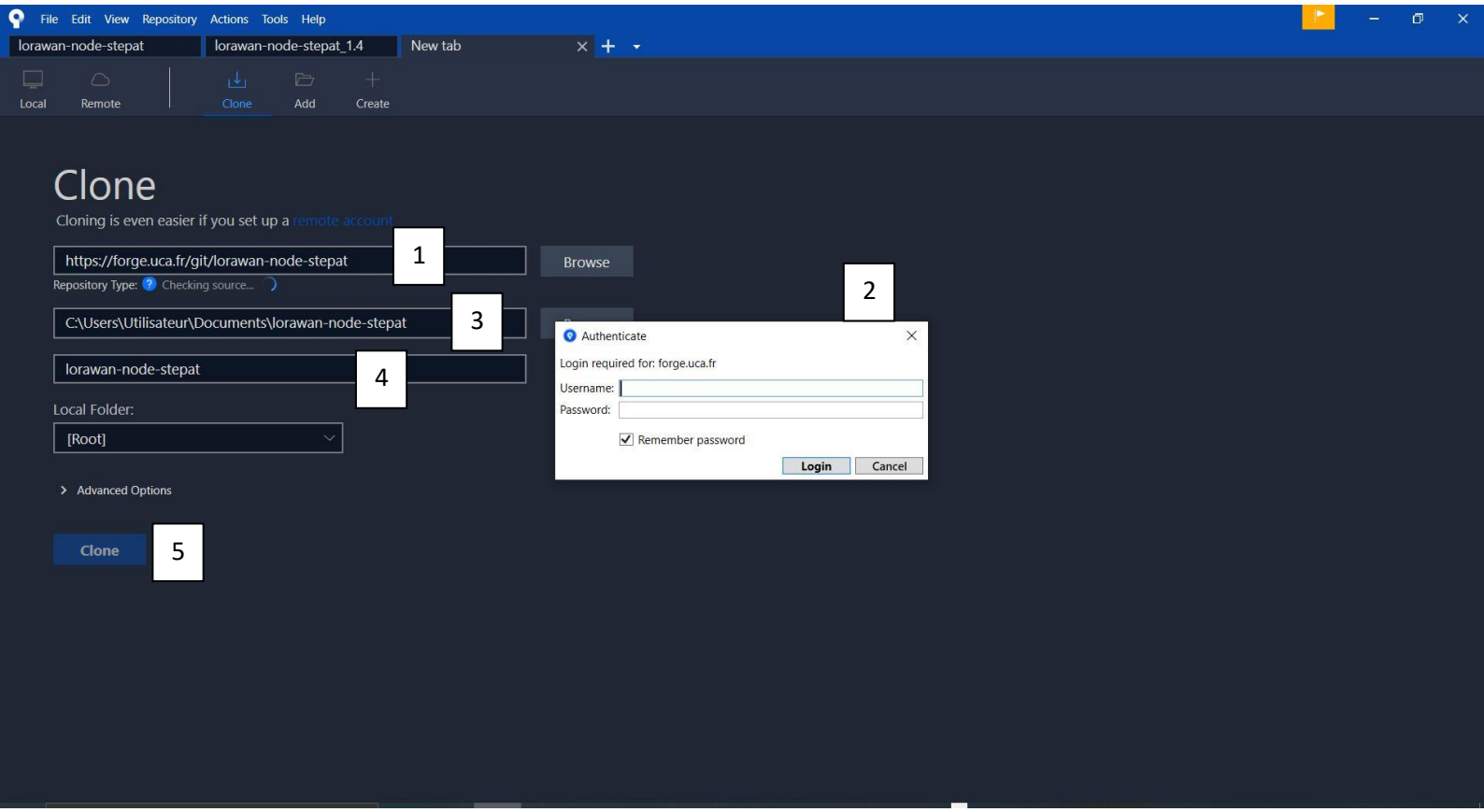

FIGURE 11 : RECUPERER LE DEPOT GIT SOURCETREE 2

On a alors notre dépôt à disposition. On peut alors cliquer sur les différents commit pour faire revenir son projet à l'état du commit sélectionné.

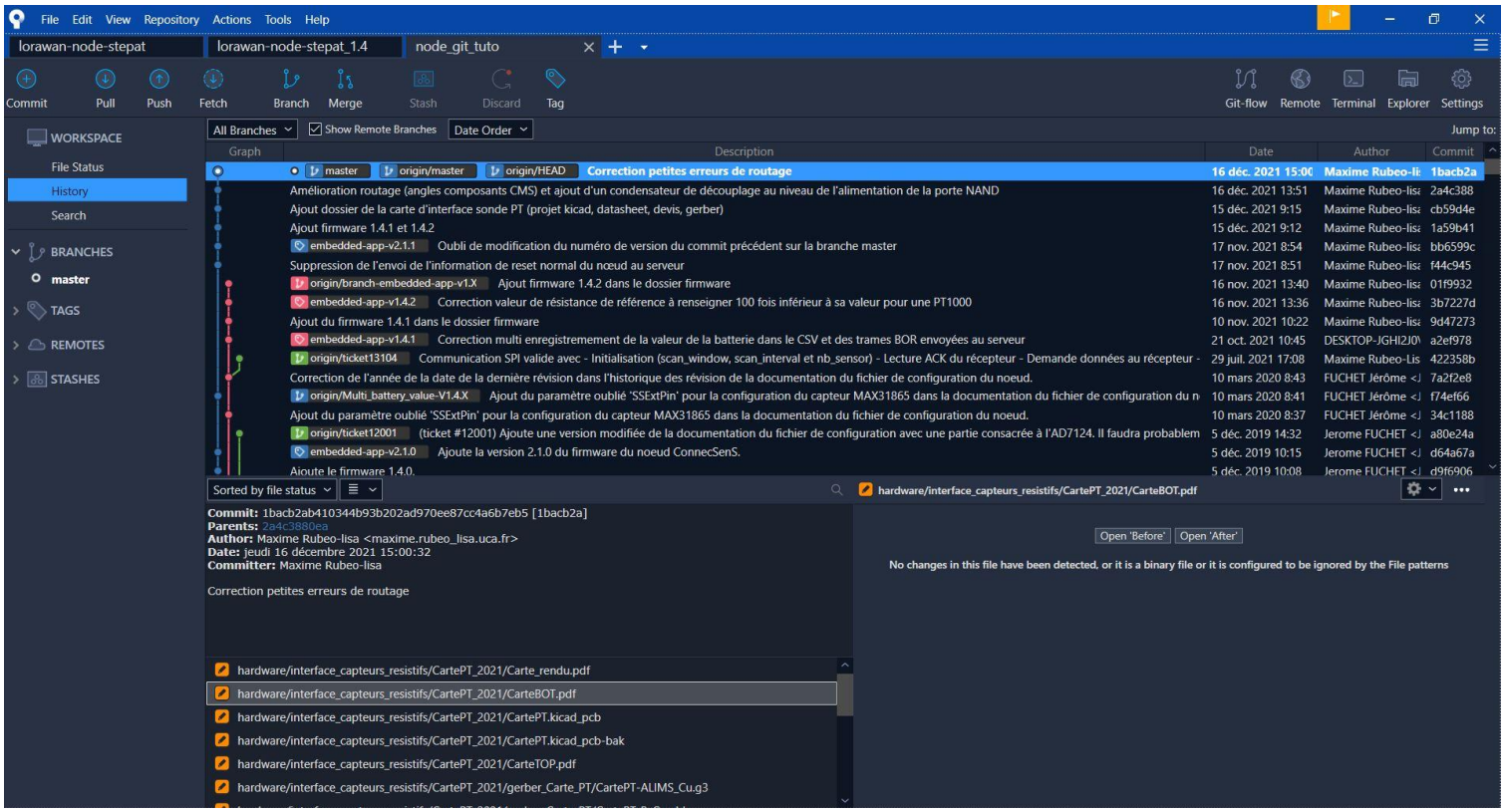

FIGURE 12 : RECUPERER LE DEPOT GIT SOURCETREE 3

On retrouve également notre dossier « node_git_tuto » à l'endroit que l'on a choisi, ici user/Documents.

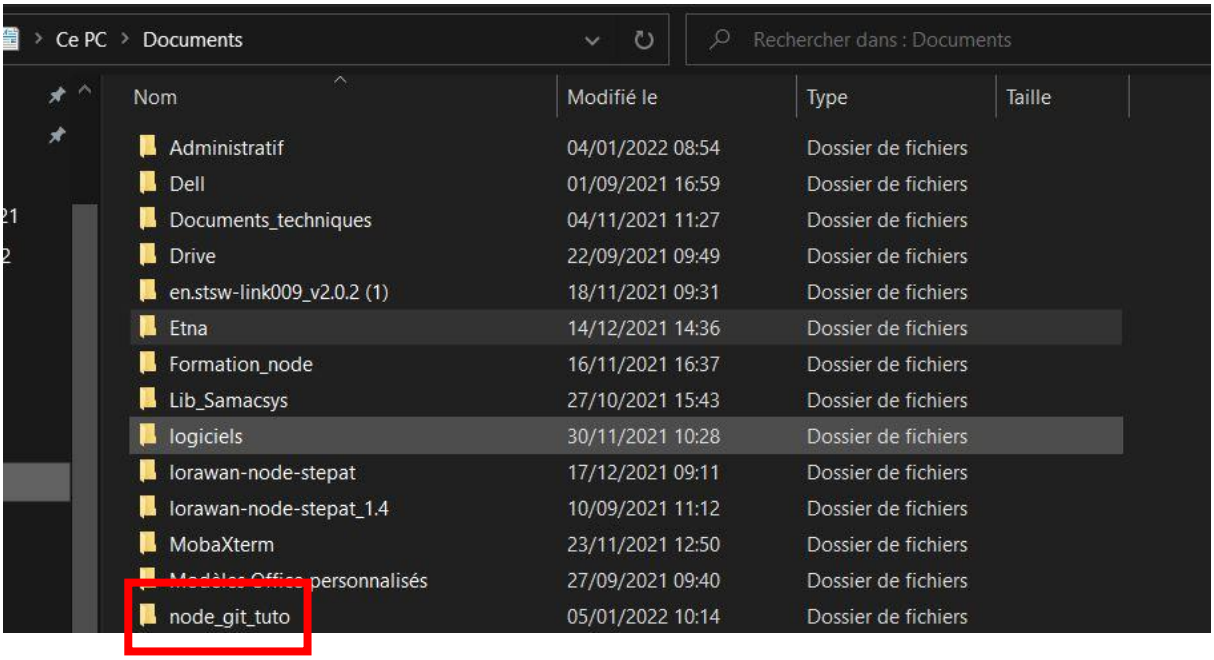

FIGURE 13 : DOSSIER GIT CREE

On a désormais accès à tout le contenu disponible sur la forge. Le software se trouve dans ce dossier dans « software\embedded\application ». On peut alors aller le modifier. Attention à bien créer sa branche pour pouvoir travailler sans casser tout le dépôt, à bien commit ses modifications (avec un commentaire pertinent à chaque fois). Une fois que l'on est sûr de nos modifications on peut alors push sur le dépôt, et faire une demande de merge pour inclure sa branche sur le projet principal.

IV. Développer sur le nœud SoLo

4.1. Prendre une bonne habitude avec GIT

Si vous voulez travailler sur le soft :

- Créer sa branche à partir de la version souhaitée (v1 ou v2)
- Faire ses modifications sur sa branche
- Modifier la version logicielle (v1.4.X ou v1.X.X, etc)
- Commit ses modifications (ne pas oublier le commentaire du commit)
- Fusionner sa branche avec la branche stable (v1 ou v2)
- Mettre le soft correspondant dans le dossier « firmwares »

4.2. Modification sur le software

Il faut noter qu'en plus du système de versionning on va garder une trace de la modification du software dans des fichiers propres au nœud.

Il faudra, pour toute modification, tenir compte de deux fichiers :

- Inc/version.h
- Changelog.txt

Le fichier version.h permet d'indiquer le numéro de version du micrologiciel. La version est composée de trois nombres séparés par un point (1.4.0 ou 2.0.1 par exemple).

La version se modifie avec les macros suivantes :

```
20  #define VERSION_MAJOR    1
21  #define VERSION_MINOR    4
22  #define VERSION_REVISION 3
```

FIGURE 14 : NUMERO DE VERSION DU SOFTWARE CONNECSENS

• **VERSION_MAJOR**: le premier nombre; est changé lorsqu'une modification importante (architecturale ou de fonctionnement) est introduite.

• **VERSION_MINOR**: le deuxième nombre; est changé lorsqu'une fonctionnalité, un pilote de capteur par exemple, est ajoutée.

• **VERSION_REVISION**: le troisième nombre. Aussi appelé numéro de patch. Est changé pour indiquer la correction d'un bogue.

Le fichier changelog.txt liste les changements notables d'une version à la suivante. Le numéro du ticket de la forge associé est indiqué.

```
1  ********************************************************************************
2  ** Version 1.4.3
3  ********************************************************************************
4  + (ticket #13926) Ajout d'un capteur fictif pour tester l'envoi d'une trame
5    contenant plusieurs datatype
6
7  ********************************************************************************
8  ** Version 1.4.2
9  ********************************************************************************
10 + (ticket #13156) Correction du facteur 100 à rensigner sur la résistance de référence
11   de la PT1000 avec le MAX31865
12
13 ********************************************************************************
14 ** Version 1.4.1
15 ********************************************************************************
16 + (ticket #13155) Conversion au format CNSSRF du reset normal du nœud commenté.
17   L'affichage dans le log de ce reset est passé de "info" à "debug"
18
19 + (ticket #13157) La partie enregistrant les données des capteurs dans le fichier CSV
20   de la fonction "InterruptHandler" a été commenté pour éviter l'enregistrement mulitple
21   de la donnée de la batterie dans le CSV.
22
23 ********************************************************************************
24 ** Version 1.4.0
25 ********************************************************************************
26 + (ticket #11421) Ajout du support de la mesure de la tempÃ©rature au moyen
27   d'un capteur RTD au platine (PTxxx) et d'un circuit d'interface basÃ© sur le
28   circuit intÃ©grÃ© MAX31865.
29
30 + (ticket #11598) Changement du systÃ¨me de logs. Adoption de celui qui a Ã©tÃ©
31   introduit dans la branche 2.x.x.
```

FIGURE 15 : FICHIER CHANGELOG.TXT

4.3. Commencer à travailler avec l'IDE Atollic TrueStudio

Au démarrage, il vous est demandé de spécifier un « workspace », qui va contenir les informations de configuration pour le projet. On peut voir sur la figure suivante que j'ai donné comme workspace un dossier de nom « Workspace_tuto ». Vous pouvez donner le nom que vous voulez à ce dossier.

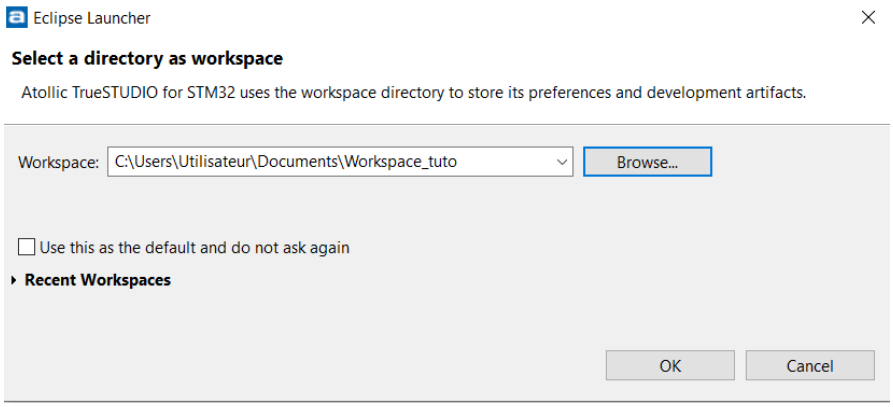

FIGURE 16 : CHEMIN DE L'ESPACE DE TRAVAIL DE L'IDE

Vous arrivez alors sur l'interface de l'IDE. Il faut alors ouvrir le projet et cliquer sur « file » puis « Open projects from file system ».

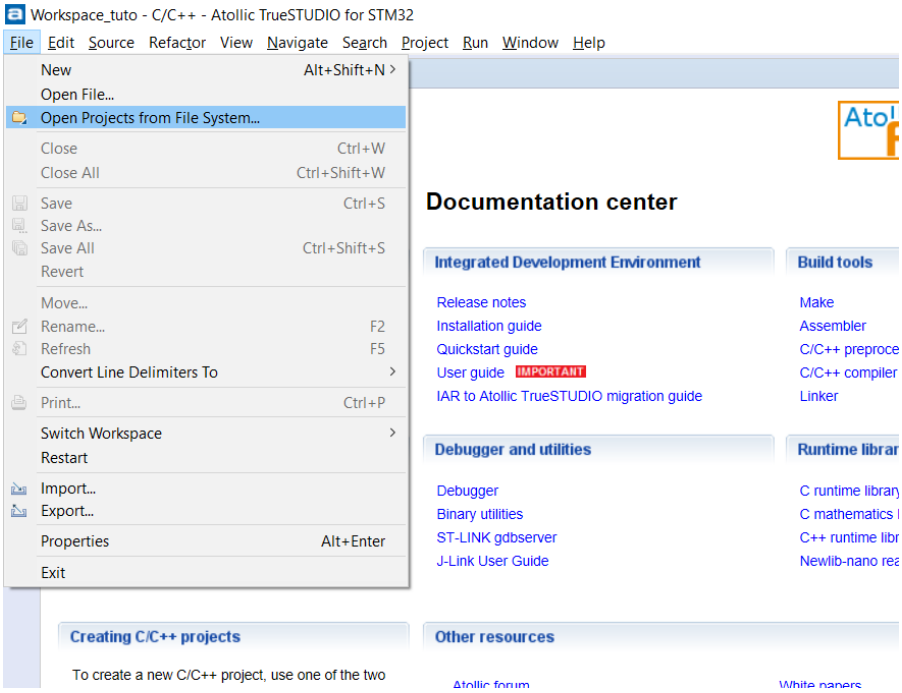

FIGURE 17 : SELECTIONNER SON PROJET 1

Il faudra alors entrer le chemin du projet, récupéré du git. Il faudra bien renseigner le chemin « software\embedded\application » depuis le dossier sur lequel vous avez récupéré le git. Vous pouvez voir le chemin que j'ai donné pour mon projet sur la figure suivante.

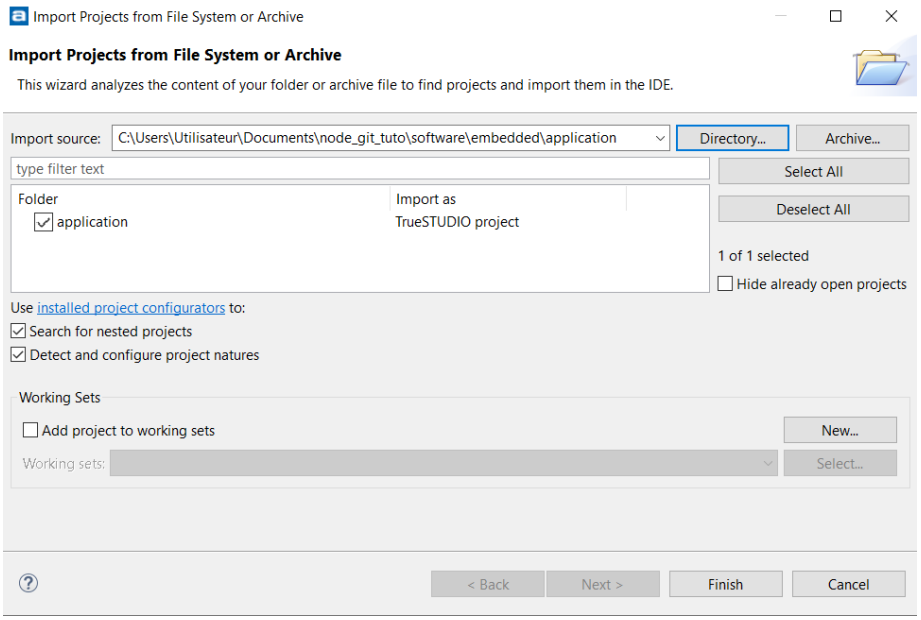

FIGURE 18 : SELECTIONNER SON PROJET 2

Cliquez sur « finish », attendez un peu et fermer la petite page de d'accueil. Vous arriverez alors sur cette interface :

Cliquez à gauche sur « restore » si le volet de navigation dans l'architecture du projet n'est pas visible

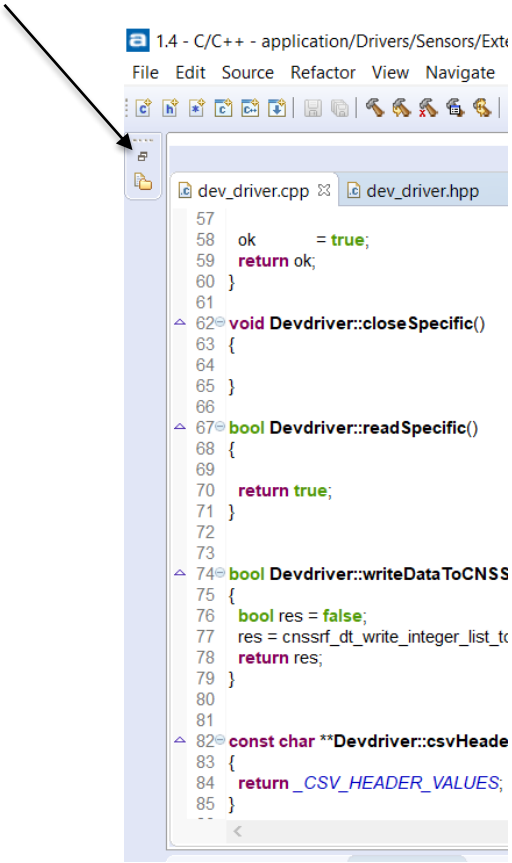

FIGURE 19 : EMPLACEMENT "RESTORE" POUR LE VOLET DE NAVIGATION DANS L'ARCHITECTURE DU PROJET

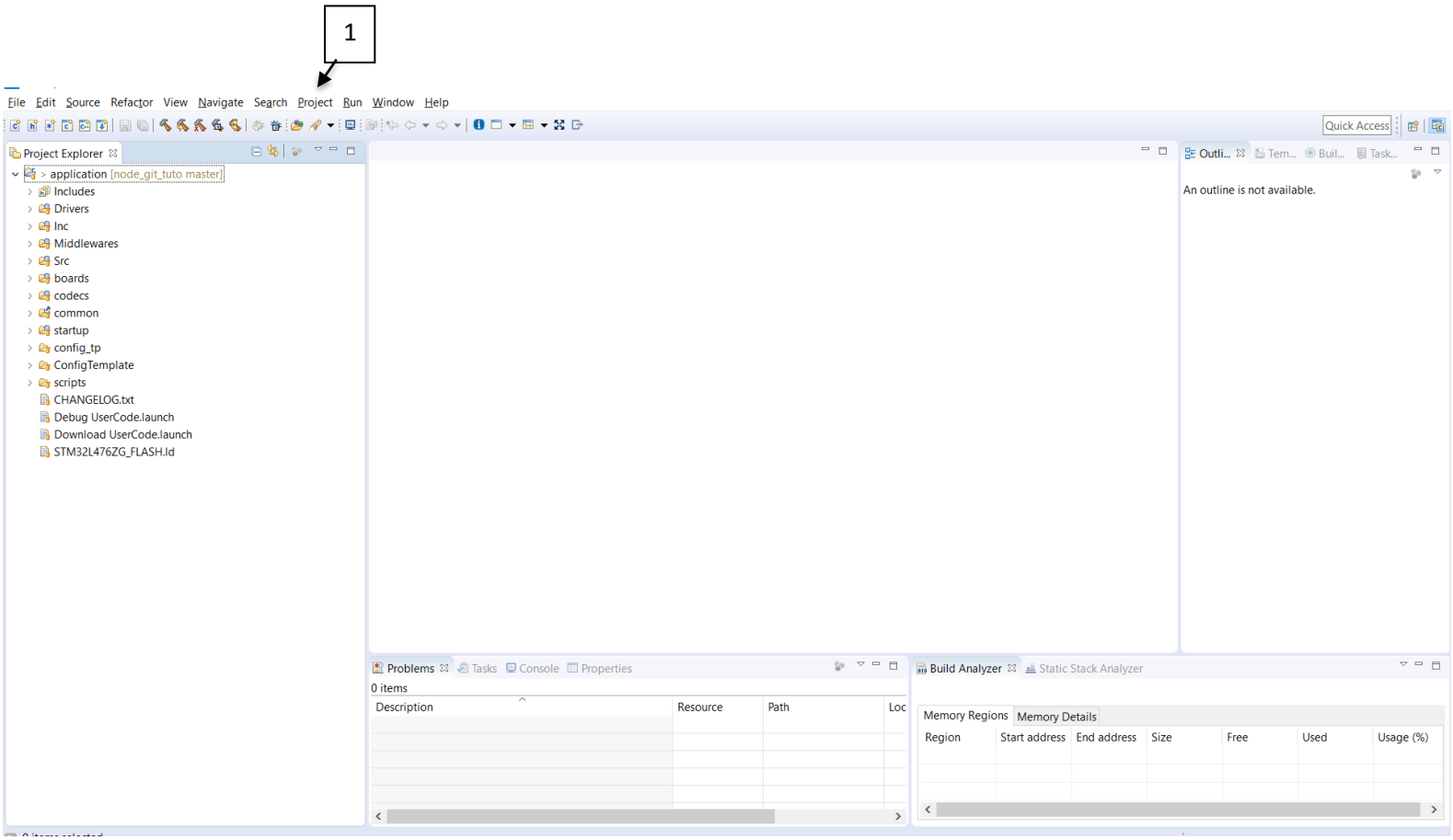

FIGURE 20 : PAGE D'ACCUEIL DE L'IDE TRUESTUDIO

Il faut bien que le builder courant soit « GNU Make Builder » pour pouvoir compiler le projet. Pour vérifier cela il faut aller dans « project » (1) puis « Build setting » -> « C/C++ build » -> Tool chain editor » :

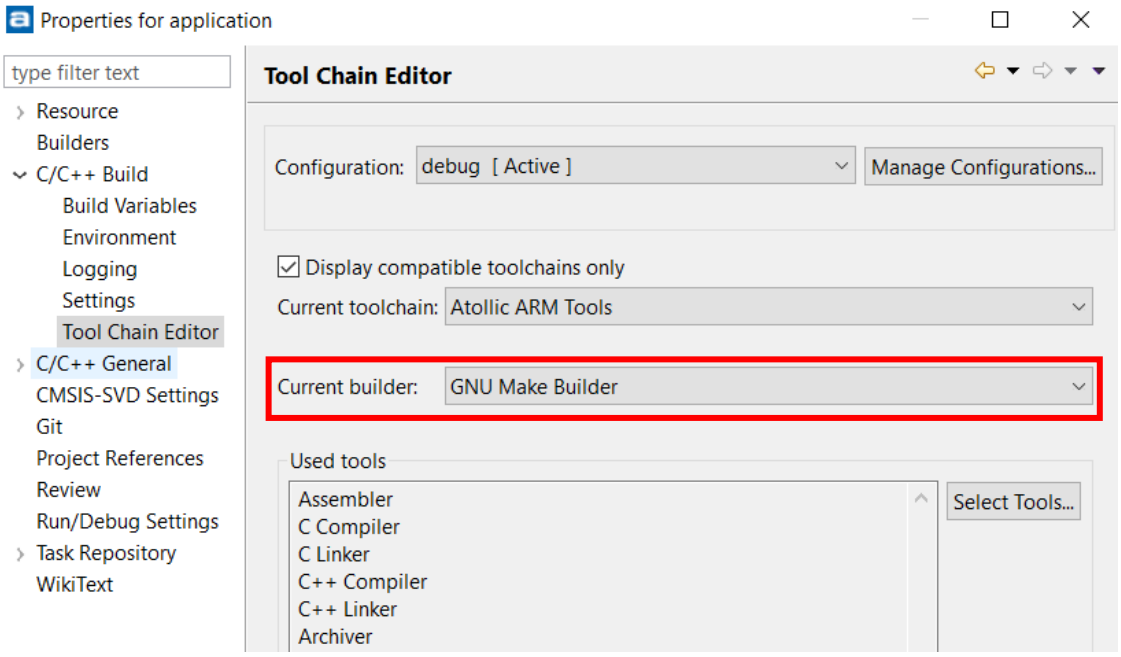

FIGURE 21 : BUILDER COURANT

Lancer la compilation

Le fichier binaire se trouve, après compilation si celle-ci réussit, dans le dossier « debug » du projet.

Son nom est application.binary, qui devra être renommé en FIRMWARE.BIN si vous voulez programmer le nœud en mettant ce fichier à la racine du nœud. Ce fichier application.binary se trouve dans le dossier « debug » du répertoire git récupéré : node_git_tuto\software\embedded\application\debug

Installer le plugin Editeur JSON

Cliquer Help

Choisir Eclipse Marketplace

- Find : json
- Choisir : json editor Plugin 1.1.2
- Install
- finish

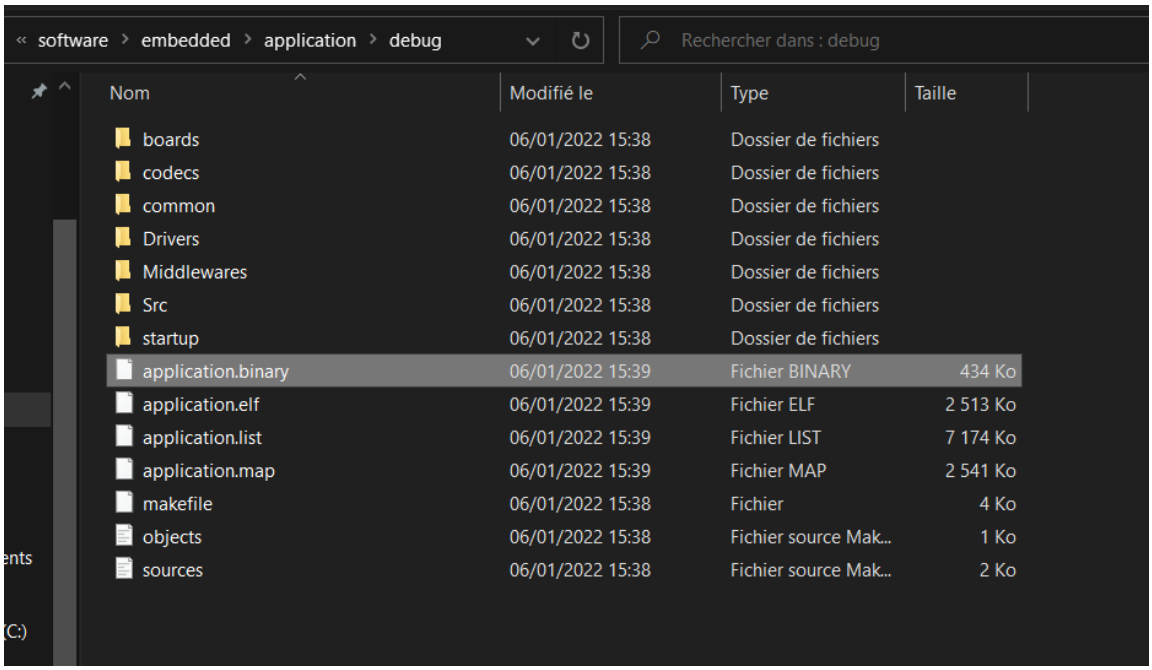

FIGURE 22 : FICHIER BINAIRE

Vous pouvez dès à présent commencer à travailler sur le soft.

4.4. Utiliser les outils de débug

Comme présenté dans la partie 2.3. Les outils de débug, il y a deux outils utilisables, le programmeur ST-Link et le convertisseur USB-TTL.

Il faut auparavant installer le logiciel STM32CubeProgrammer disponible sur le site ST (https://www.st.com/en/development-tools/stm32cubeprog.html)

4.3.1. Convertisseur USB-TTL

Pour pouvoir lire les logs sur une invite de commande, on va utiliser un outil spécifique : un convertisseur USB-TTL (figure 4).

Pour pouvoir l'utiliser il faut connecter la masse de l'adaptateur (fil noir) à une masse du nœud et le fil Rx de l'adaptateur (fil blanc) à la broche USART_TX du nœud.

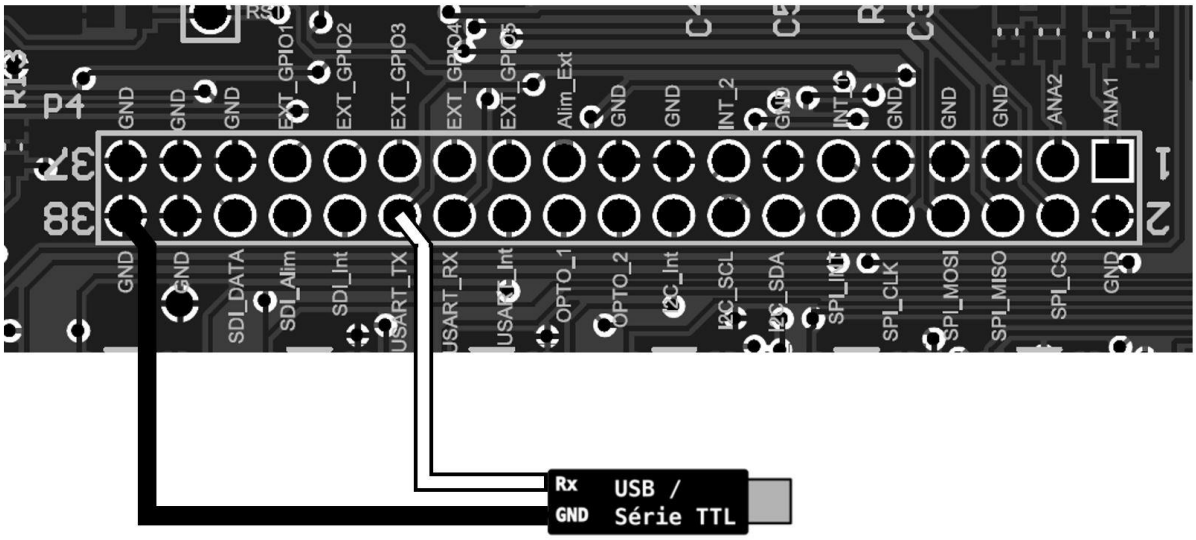

FIGURE 23 : CONNECTIQUE USB-TTL

Il faut ensuite changer le fichier de configuration du nœud (config.json) dans le dossier « env » du nœud. Pour cela connecter en usb votre nœud. Un dossier apparait. Vous trouverez la configuration dans « env », puis ouvrez le fichier config.json et modifier la partie « logs » en y ajoutant « "serial" : "EXT_USART" » :

```
[...],

"logs": {
    "defaultLevel": "DEBUG",
    "serial":       "EXT_USART"
},
```

FIGURE 24 : CONFIGURATION LECTURE DES LOGS

```
A copier :
"logs": {
            "defaultLevel": "DEBUG",
```

```
        "serial": "EXT_USART"
    },
```

Pour pouvoir lire les logs du nœud nous allons utiliser un script python qui utilise les scripts de décodage du serveur. Il faut télécharger sur son ordinateur python3 ainsi que le dossier « nodelogparser » sur le git « lorawan-server-app » (dans /tools/python).

Une fois ce dossier télécharger, lancer le script python « nodelogparser.py » :

```
python nodelogparser_serial.py
```

FIGURE 25 : COMMANDE POUR LANCER LE SCRIPT DE LECTURE DES LOGS SUR UNE INVITE DE COMMANDE WINDOWS

J'ai une erreur en lançant le script : il ne trouve pas « import tools_common »

Les logs s'afficheront directement dans l'invite de commande une fois le nœud en fonctionnement :

```
2022-02-14 13:54:00|DEBUG|connecsens|1077|Internal power: ENABLE
2022-02-14 13:54:00|DEBUG|connecsens|1112|External interruptions power: DISABLE
2022-02-14 13:54:00|DEBUG|connecsens|302|Opening datalog file 'env/datalogs/434E535301E31302-sensors.cnssrf.dlog'...
2022-02-14 13:54:00|DEBUG|datalog_cnssrf|227|Datalog indexes file is: env/datalogs/434E535301E31302-sensors.cnssrf.dlog.
indexes
2022-02-14 13:54:00|DEBUG|connecsens|305|CNSSRF datalog file 'env/datalogs/434E535301E31302-sensors.cnssrf.dlog' has bee
n opened.
2022-02-14 13:54:00|DEBUG|connecsens|938|Node reseted because powered on or BOR.
2022-02-14 13:54:00|INFO |connecsens|444|CNSSRF payload: C18400781B9D290F3811242BD07BD82B42842093F901
2022-02-14 13:54:00|DEBUG|connecsens|452|CNSSRF meta data: 012015400561036105610761
2022-02-14 13:54:00|DEBUG|sensor|557|TempHumi: Opening...
2022-02-14 13:54:00|DEBUG|sensor|558|TempHumi: Opened.
2022-02-14 13:54:00|DEBUG|sensor|586|TempHumi: Reading...
2022-02-14 13:54:00|INFO |sensor|590|TempHumi: Reading done.
2022-02-14 13:54:00|DEBUG|sensor|577|TempHumi: Closed.
2022-02-14 13:54:00|DEBUG|sensor|557|Pressure: Opening...
2022-02-14 13:54:00|DEBUG|sensor|558|Pressure: Opened.
2022-02-14 13:54:00|DEBUG|sensor|586|Pressure: Reading...
2022-02-14 13:54:00|INFO |sensor|590|Pressure: Reading done.
2022-02-14 13:54:01|DEBUG|sensor|577|Pressure: Closed.
2022-02-14 13:54:01|DEBUG|sensor|557|Lumi: Opening...
2022-02-14 13:54:01|DEBUG|sensor|558|Lumi: Opened.
2022-02-14 13:54:01|DEBUG|sensor|586|Lumi: Reading...
2022-02-14 13:54:01|INFO |sensor|590|Lumi: Reading done.
2022-02-14 13:54:01|DEBUG|sensor|577|Lumi: Closed.
2022-02-14 13:54:01|INFO |sensor|401|Lumi: Sensor's state file does not exist; use default state.
2022-02-14 13:54:01|INFO |connecsens|444|CNSSRF payload: C18300781B9D290F3811242BD07BD80B2346A9422502930B05411223769AA85
104E8681A230FAD26D008D200
```

FIGURE 26 : LOGS DU NŒUD SUR UNE INVITE DE COMMANDE WINDOWS

4.3.2. Programmeur ST-Link/V2-Isol

Le module ST-Link va servir à programmer le nœud avec un nouveau firmware sans avoir à l'ajouter à la racine du dossier du nœud (fichier nommé FIRMWARE.BIN).

Le nœud dispose d'une interface JTAG. C'est sur cette dernière que nous allons venir connecter la sonde.

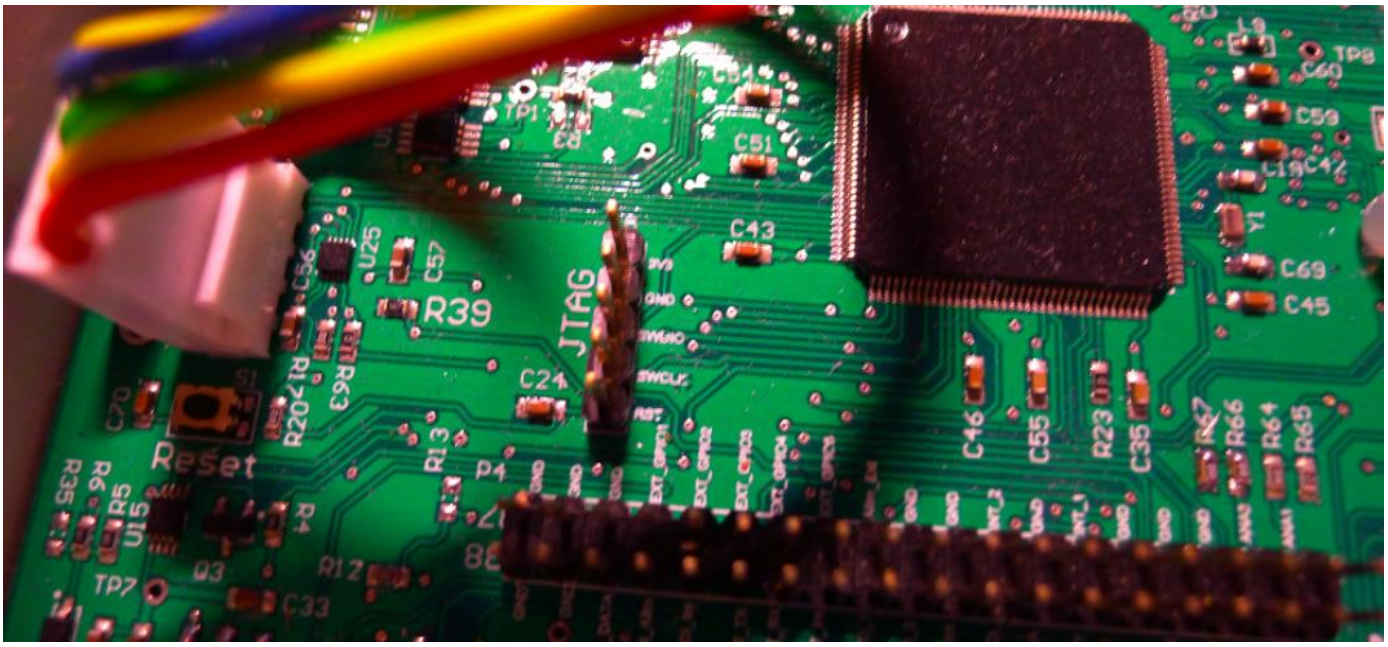

FIGURE 27 : PORT JTAG DU NŒUD

Le brochage du connecteur JTAG du nœud est le suivant :

FIGURE 28 : BROCHAGE DU CONNECTEUR JTAG

C'est un connecteur mâle de pas 2.54mm de 5 contacts sur une rangée.

Voici le schéma de câblage entre une sonde ST-Link/V2-Isol et le port JTAG du nœud (Le câblage change si vous utilisé la version ST-Link/V2 simple

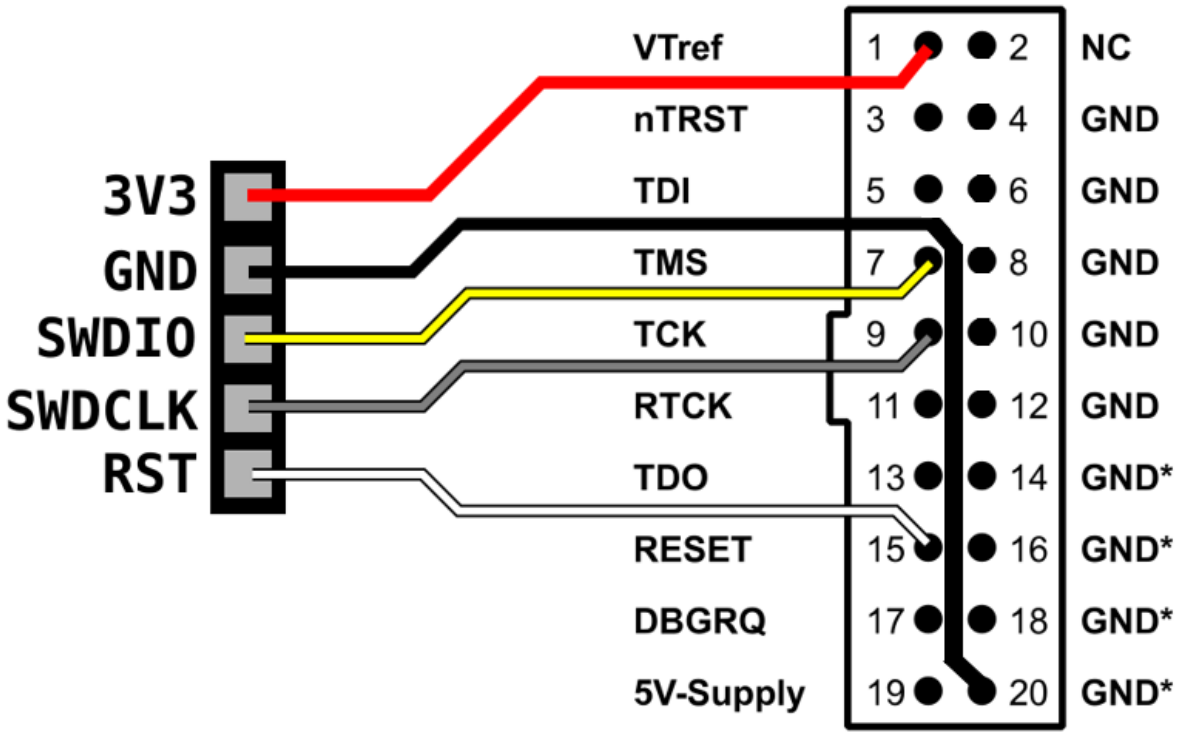

FIGURE 29 : CONNECTIQUE ENTRE LA BORNE JTAG DU NŒUD ET LE PROGRAMMEUR

Avant de programmer le nœud il fut d'assurer que le projet « application » est bien en « build configuration » debug. Pour vérifier cela on fait clic droit sur le projet, puis « build configuration », « Set active » et enfin « debug »

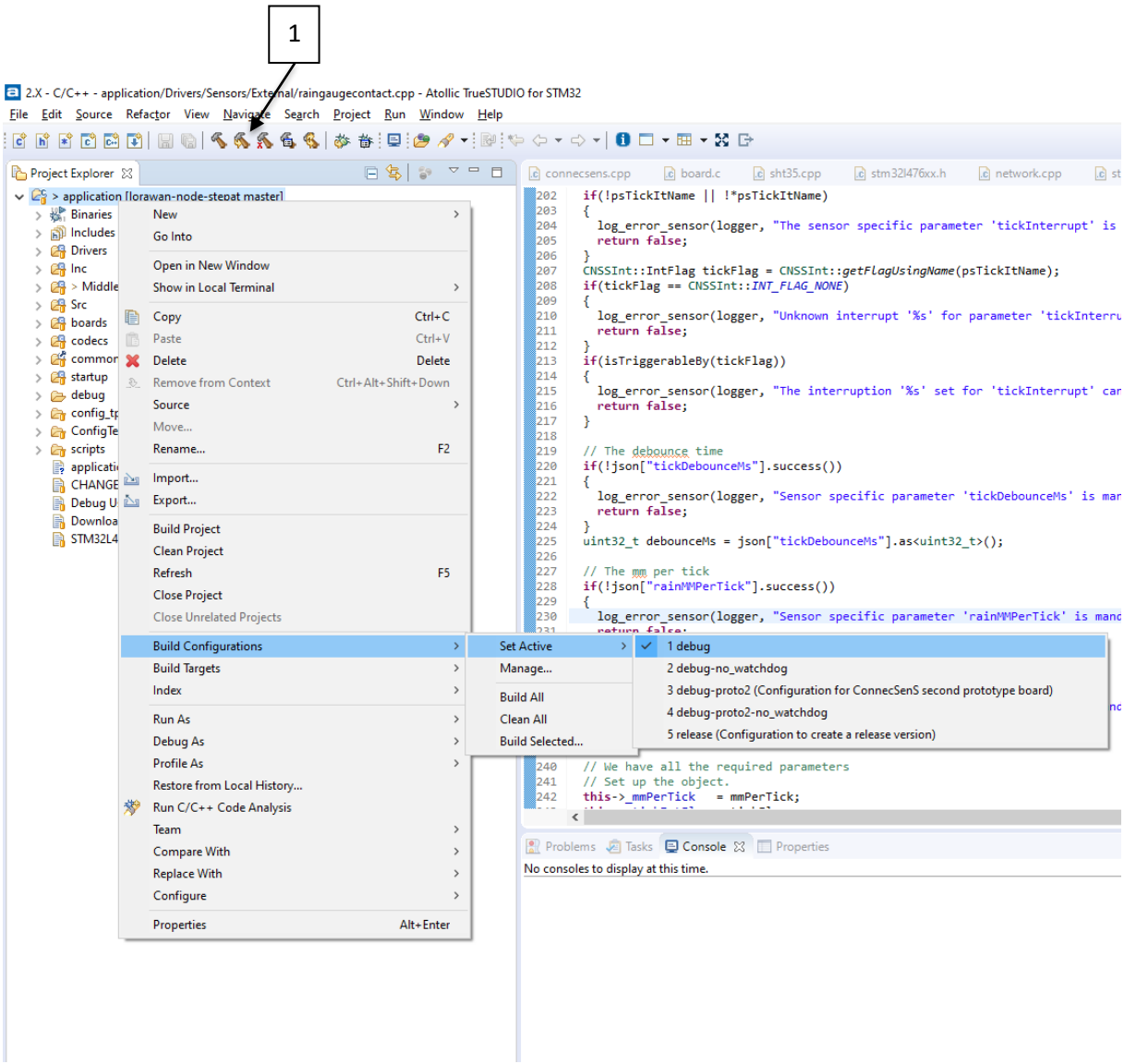

FIGURE 30 : DEBUG CONFIGURATION

Il faut également « build » le projet (icône 1). Il faut ensuite alimenter le nœud par sa batterie. Vous pouvez l'alimenter par USB et par batterie. Il faudra alors retirer l'USB après avoir programmer le nœud avec le nouveau firmware pour lancer le nœud et voir les log s'afficher.

La programmation du nœud peut être lancée par une « debug configuration » ou par un « run external tool »

Nous allons utiliser « run external tool » dans ce guide. Nous allons créer une nouvelle instance pour le programmeur. Aller dans « run », puis « external tools » et « external tools configurations ».

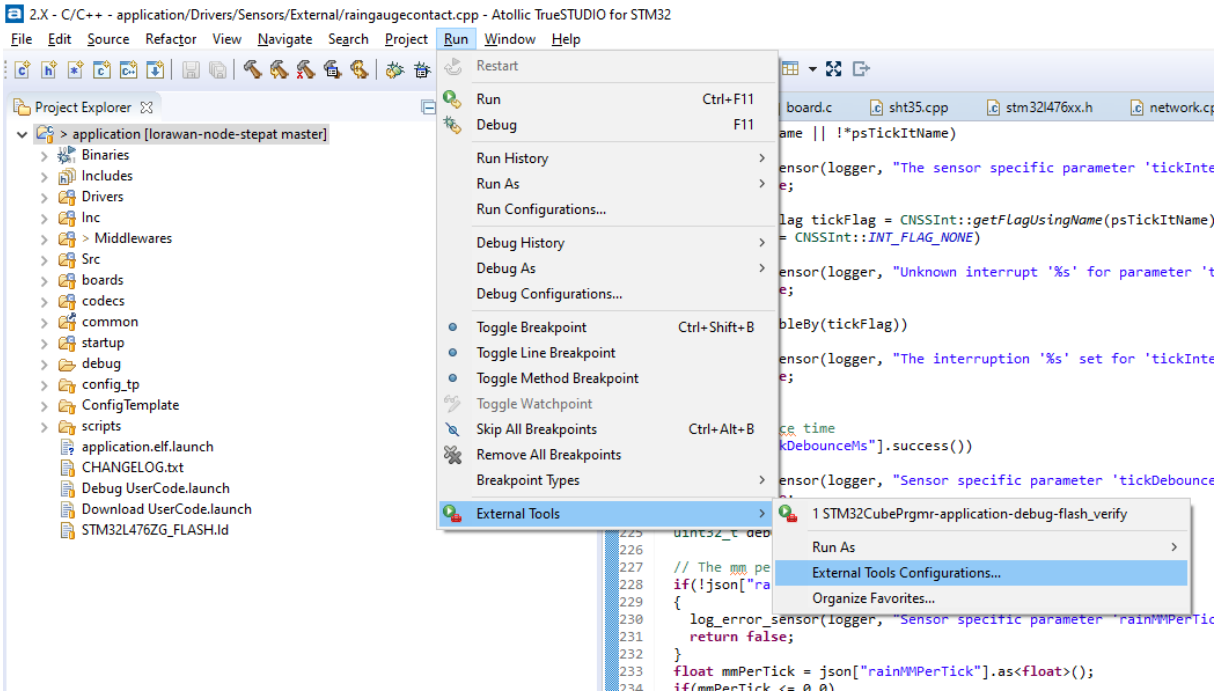

FIGURE 31 : RUN EXTERNAL TOOLS

Cliquez sur “Program”

Faite « new launch configuration ». Il va ensuite falloir donner un nom à la configuration, puis le chemin de « STM32_Programmer_CLI.exe » (pour ma part il se trouve au chemin « C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\bin\STM32_Programmer_CLI.exe » mais cela peut varier suivant où vous l’installez).

Vous devez également donner le chemin du répertoire où se trouve le fichier binaire du projet (« C:\Users\Utilisateur\Documents\lorawan-node-stepat\software\embedded\application\debug » sur mon projet).

Dans la section « argument » recopiez le contenu suivant :

-c port=SWD mode=UR reset=hwRst --download ${workspace_loc:/application/debug/application.elf} --verify --quietmode –hardRst

Enregistrez les modifications et lancer l’outil de programmation. Si vous aviez connecter le nœud à votre ordinateur avec le câble USB, à la fin de la programmation débranchez l’USB qui alimente le nœud pour le démarrer. N’oubliez pas de brancher la batterie. Les logs vont alors apparaitre sur l’invite de commande si vous avez lancé le script python (voir partie Convertisseur USB-TTL).

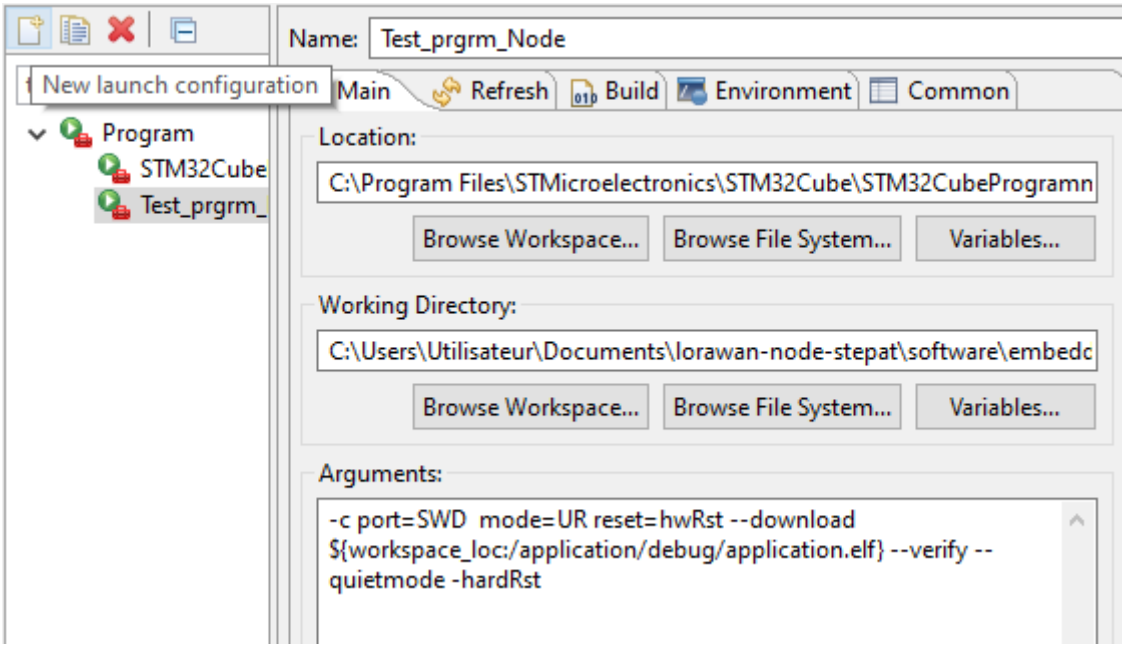

FIGURE 32 : CONFIGURATION DU RUN EXTERNAL TOOLS

V. Ajouter un nouveau driver

5.1. Création des fichiers du driver

Dans cette partie, je vais vous montrer comment ajouter un nouveau driver au nœud. Je vais me placer dans le cas où vous voulez seulement ajouter un driver simple. Si vous souhaitez créer un driver générique, vous allez devoir développer des classes génériques. Dans ce cas, je vous conseille fortement de suivre le TP de Jérôme Fuchet.

Je considère que vous avez récupéré le software du nœud. La figure suivante vous montre l'architecture du projet.

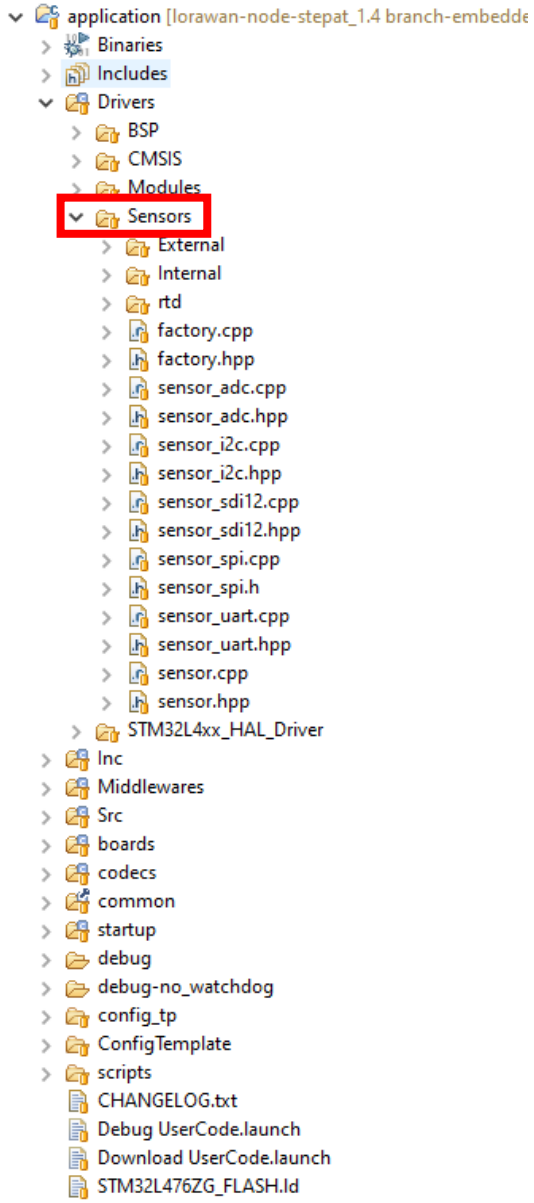

FIGURE 33 : ARCHITECTURE SENSORS

Vous allez ajouter votre driver dans la partie *Sensors -> External*. Le driver que je vais créer se nomme *dev_driver*. Ce driver ne fera rien de matériel (pas de communication I²C, SPI, etc). Si votre driver utilise un de ces protocoles, je vous conseille de vous inspirer d'un driver déjà existant. Le nom de ce driver commencera par convention par une minuscule. Si le nom est composé, vous pouvez utiliser un « _ » (underscore, le tiret du « bas ») ou séparer mettre une majuscule au début de chaque mot (*DevDriver* par exemple).

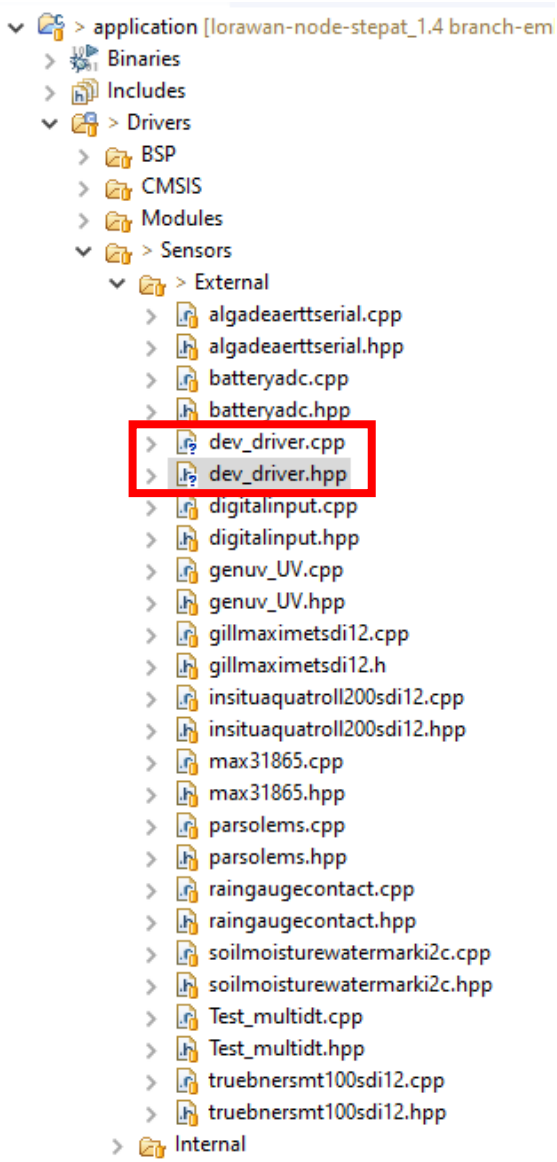

FIGURE 34 : EMPLACEMENT FICHIERS DRIVER

Le plus simple pour être sûr de ne pas oublier une fonction nécessaire est de partir d'un driver déjà existant. Il faudra alors renommer toutes les instances nécessaires. Je vais vous montrer comment procéder.

Pour ce driver, je suis parti d'un driver test que j'avais utilisé pour tester un nouveau format de donnée. Toutes les instances portent donc un nom spécifique (ici *Multidt*). Nous allons donc changer tous ces noms pour qu'ils conviennent à notre driver. Nous utiliserons le nom *Dev_driver*. Je vous conseille

d'utiliser un nom explicite. Par convention, la classe commencera par une majuscule. Les méthodes et variables commenceront par une minuscule. Vous trouverez sur la figure suivante 2 fichiers. Dans le fichier .hpp qui contient toutes vos fonctions et variables pour votre code, vous allez devoir renommer toutes instances. Nous changerons donc les noms de la classe, du constructeur et du destructeur (carré rouge) ainsi que les noms des directives préprocesseur (carré vert)

dev_driver.cpp | dev_driver.hpp

```
1  /*
2   * Test to send a list of datas
3   *
4   * @author Maxime Rubeo-Lisa
5   * @date   2022
6   */
7  #ifndef SENSORS_EXTERNAL_MULTIDT_HPP_
8  #define SENSORS_EXTERNAL_MULTIDT_HPP_
9
10 #include "config.h"
11 #ifdef USE_SENSOR_MULTIDT
12 #include "sensor.hpp"
13
14 class Multidt : public Sensor
15 {
16 public:
17   static const char *TYPE_NAME;  ///< The sensor's type name.
18   Multidt();
19   ~Multidt();
20   static Sensor *getNewInstance();
21   const char    *type();
22
23
24 private:
25
26   bool openSpecific();
27   void closeSpecific();
28   bool readSpecific();
29   bool jsonSpecificHandler(          const JsonObject& json);
30   bool writeDataToCNSSRFDataFrameSpecific(CNSSRFDataFrame  *pvFrame);
31
32   const char **csvHeaderValues();
33   int32_t     csvDataSpecific(char *ps_data, uint32_t size);
34
35   uint16_t                Nb_data;
36   uint16_t                _Data[10];
37   static const char *_CSV_HEADER_VALUES[];  ///< The CSV header values.
38 };
39
40 #endif // USE_SENSOR_MULTIDT
41 #endif /* SENSORS_EXTERNAL_MULTIDT_HPP_ */
42
```

FIGURE 35 : MODIFICATION NOM FICHIER HPP

Vous trouverez sur la figure suivante le résultat de ces changements.

dev_driver.cpp | dev_driver.hpp

```
1  /*
2   * Test to send a list of datas
3   *
4   *  @author Maxime Rubeo-Lisa
5   *  @date   2022
6   */
7  #ifndef SENSORS_EXTERNAL_DEVDRIVER_HPP_
8  #define SENSORS_EXTERNAL_DEVDRIVER_HPP_
9
10 #include "config.h"
11 #ifdef USE_SENSOR_DEVDRIVER
12 #include "sensor.hpp"
13
14 class Devdriver: public Sensor
15 {
16 public:
17   static const char *TYPE_NAME;  ///< The sensor's type name.
18   Devdriver();
19   ~Devdriver();
20   static Sensor *getNewInstance();
21   const char    *type();
22
23
24 private:
25
26   bool openSpecific();
27   void closeSpecific();
28   bool readSpecific();
29   bool jsonSpecificHandler(           const JsonObject& json);
30   bool writeDataToCNSSRFDataFrameSpecific(CNSSRFDataFrame  *pvFrame);
31
32   const char **csvHeaderValues();
33   int32_t     csvDataSpecific(char *ps_data, uint32_t size);
34
35   uint16_t                Nb_data;
36   uint16_t                _Data[10];
37   static const char *_CSV_HEADER_VALUES[];  ///< The CSV header values.
38 };
39
40 #endif // USE_SENSOR_DEVDRIVER
41 #endif /* SENSORS_EXTERNAL_DEVDRIVER_HPP_ */
42
```

FIGURE 36 : NOM APRES MODIFICATION

Ces mêmes changements doivent être fait sur votre .cpp

```
7  #include "dev_driver.hpp"
8  #ifdef USE_SENSOR_DEVDRIVER
9  #include <string.h>
10 #include "board.h"
11
12
13 CREATE_LOGGER(Devdriver);
14 #undef  logger
15 #define logger  Devdriver
16
17
18 const char *Devdriver::TYPE_NAME = "Devdriver";
19
20 const char *Devdriver::_CSV_HEADER_VALUES[] =
21 {
22    "Value_2", "Value_2", "Value_3", "Value_4", NULL
23 };
24
25
26
27 Devdriver::Devdriver() : Sensor(POWER_NONE, POWER_NONE, FEATURE_BASE)
28 {
29
30 }
```

FIGURE 37 : NOM FICHIER CPP

Attention à inclure le bon header (.hpp) (voir carré rouge sur la figure précédente).

Vous remarquerez que le programme est grisé. C'est normal, nous avons changé les directives préprocesseur. Ces instances ne sont pas prises en compte par le programme, c'est comme si elles n'étaient pas déclarées. Pour les déclarer, vous allez devoir modifier 2 fichiers supplémentaires.

Dans le fichier *factory.hpp*, inclure votre header :

*factory.cpp | config.h | versic

```
9  #include "simulation.hpp"
10 #include "sht35.hpp"
11 #include "opt3001.hpp"
12 #include "lps25.hpp"
13 #include "lis3dh.hpp"
14 #include "raingaugecontact.hpp"
15 #include "soilmoisturewatermarki2c.hp
16 #include "insituaquatroll200sdi12.hpp"
17 #include "batteryadc.hpp"
18 #include "gillmaximetsdi12.h"
19 #include "algadeaerttserial.hpp"
20 #include "truebnersmt100sdi12.hpp"
21 #include "digitalinput.hpp"
22 #include "max31865.hpp"
23 #include "Test_multidt.hpp"
24 #include "genuv_UV.hpp"
25 #include "parsolems.hpp"
26 #include "dev_driver.hpp"
27
```

FIGURE 38 : AJOUT HEADER AU FICHIER FACTORY.HPP

Dans ce même fichier ajouter l'instance que vous venez de créer (*USE_SENSOR_DEVDRIVER* et le nom des classes *Devdriver*)

```
#ifdef USE_SENSOR_PARSOLEMS_UV
  { PARsolems::TYPE_NAME,          &PARsolems::getNewInstance
#endif
#ifdef USE_SENSOR_DEVDRIVER
  { Devdriver::TYPE_NAME,          &Devdriver::getNewInstance          },
#endif
  { NULL, NULL }
};
```

FIGURE 39 : AJOUT INSTANCE FICHIER FACTORY.HPP

Dans le fichier *config.h* ajouter votre directive à la suite des autres :

factory.cpp | config.h | version.h | CHANGELOG

```
3  /* Configuration file for ConnecSenS System **********
4  /*****************************************************
5  /*****************************************************
6
7
8  #define CURRENT_YEAR_MIN 2019 // The current year minimu
9
10 #define BATT_VOLTAGE_FULL_V       4.2
11 #define BATT_VOLTAGE_EMPTY_V      2.5
12 #define BATT_VOLTAGE_LOW_THRESHOLD_V 3.2 // Volts
13
14
15 #define USE_SENSOR_RAIN_GAUGE_CONTACT
16 #define USE_SENSOR_SOIL_MOISTURE_WATERMARK_I2C
17 #define USE_SENSOR_INSITU_AQUATROLL_200_SDI12
18 #define USE_SENSOR_BATTERY_ADC
19 #define USE_SENSOR_GILL_MAXIMET_SDI12
20 #define USE_SENSOR_ALGADE_AERTT_SERIAL
21 #define USE_SENSOR_TRUEBNER_SMT100_SDI12
22 #define USE_SENSOR_DIGITAL_INPUT
23 #define USE_SENSOR_MAX31865
24 #define USE_SENSOR_MULTIDT
25 #define USE_SENSOR_GENUV_UV
26 #define USE_SENSOR_PARSOLEMS_UV
27 #define USE_SENSOR_DEVDRIVER
28
```

FIGURE 40 : AJOUT DIRECTIVE PREPROCESSEUR FICHIER CONFIG.H

Une fois que vous avez fait tout cela, vous verrez que votre code dans vos fichiers du driver n'est plus grisé, il sera alors pris en compte lors de la compilation.

Je vais passer à la définition des différentes fonctions nécessaires au bon fonctionnement du driver

5.2. Fonctions nécessaires au fonctionnement du driver

Dans votre fichier header (.hpp), plusieurs fonctions sont nécessaires au bon fonctionnement du driver.

```
14  class Devdriver : public Sensor
15  {
16  public:
17    static const char *TYPE_NAME;  ///< The sensor's type name.
18    Devdriver();
19    ~Devdriver();
20    static Sensor *getNewInstance();
21    const char   *type();
22
23
24  private:
25
26    bool openSpecific();
27    void closeSpecific();
28    bool readSpecific();
29    bool jsonSpecificHandler(          const JsonObject& json);
30    bool writeDataToCNSSRFDataFrameSpecific(CNSSRFDataFrame  *pvFrame);
31
32    const char **csvHeaderValues();
33    int32_t     csvDataSpecific(char *ps_data, uint32_t size);
34    static const char *_CSV_HEADER_VALUES[];  ///< The CSV header values.
35
36
37    uint16_t              Nb_data;
38    uint16_t              _Data[10];
39  };
```

FIGURE 41 : CONTENU HEADER DU DRIVER

La classe : Devdriver est un objet qui dérive de la classe *Sensor*. Vous pouvez utiliser une autre classe plus adéquat dans le cas où vous avez besoin de fonctions particulières (*SensorI2C* si vous avez besoin de fonctions I²C par exemple).

Les méthodes :

- Devdriver() : le constructeur, qui va permettre d'initialiser les variables utilisées dans le programme.
- ~Devdriver() : le destructeur, fait l'inverse du constructeur
- *getNewInstance() : Permet d'obtenir une nouvelle instance du capteur
- *type : Permet de définir le type du capteur, c'est grâce à cette instance que l'on va pouvoir identifier le capteur dans le fichier de configuration. Elle permet de renvoyer le nom du capteur défini dans TYPE_NAME.
- openSpecific() : Permet de créer un nouvel objet, pour le SPI par exemple, *return _spi.open()* pour signifier au programme que l'on va utiliser les fonctions spi. Ne rien faire si n'est pas utilisé (*return true*)
- closeSpecific() : permet de fermer l'objet ouvert, *_spi.close()* si je reprends l'exemple précédent. Ne rien faire si n'est pas utilisé.
- readSpecific() : permet de lire les valeurs renvoyées par le capteur.
- jsonSpecificHandler() : Permet de spécifier des paramètres dans le fichier JSON. Par défaut, seul 2 paramètres JSON sont utilisé et n'ont pas besoin d'être déclaré. Ces 2 paramètres sont « name » qui contient le nom que l'on va donner à notre capteur et « type », qui doit correspondre au nom du capteur dans le programme pour qu'il puisse être utilisé.

- writeDataToCNSSRFDataFrameSpecific() : Permet d'écrire les données du capteur au format CNSSRF
- *_CSV_HEADER_VALUES : permet de créer les noms des colonnes pour le fichier CSV
- **csvHeaderValues() : retourne nos noms de colonnes pour le fichier csv
- csvDataSpecific() : Permet d'écrire les données dans le fichier csv

Les variables :

- *TYPE_NAME : Le nom du capteur, nécessaire pour le fichier de configuration du nœud
- Nos autres variables nécessaire pour notre driver, dans cette exemple Nb_data et _Data[10]. Les variables déclarées ici doivent être initialisées dans le constructeur

Il ne faudra pas oublier d'ajouter au git les fichiers créés, de modifier le numéro de version du firmware et d'ajouter les modifications au changelog.