Développement d'un objet connecté - Qt

Sommaire

1. Introduction . . . 2
2. Objectif de l'activité . . . 3
3. The Things Network . . . 3
   3.1. Présentation . . . 3
   3.2. Installation d'une passerelle . . . 5
4. Administration de la passerelle et des applications . . . 6
   4.1. Administration . . . 7
   4.2. Ajout d'une application . . . 7
5. Ajout d'un objet connecté . . . 9
   5.1. Enregistrement d'un nouvel objet connecté . . . 9
   5.2. Rappel : Méthodes d'activation . . . 11
   5.3. Configuration du Device Extended Unique Identifier (`DevEUI`) . . . 13
   5.4. Envoyer et recevoir . . . 15
6. Serveur d'application . . . 20
   6.1. Installation du module `QtMQTT` . . . 22
   6.2. Exemple `simpleclient` . . . 23
   6.3. Notre application . . . 29
7. Conclusion . . . 40

1. Introduction

Les objets connectés ou l'**Internet des Objets** (*IoT : Internet of Things*) sont des termes très utilisés de nos jours même si bien des personnes ont encore du mal les définir.

Au sens large, un objet connecté est un objet communicant. Ils sont à la fois des émetteurs, des capteurs d'informations qui émettent, mais également qui peuvent interagir avec d'autres machines (M2M) comme des serveurs ou d'autres objets connectés. Les données générées par ces objets sont capitalisées dans des centres de données (*data centers*) et de nouveaux usages de cette donnée apparaissent autour de la santé, du sport, des produits financiers par exemple.

Les objets connectés connaissent une croissance exponentielle. Des estimations indiquent qu'ils seraient près de 15 milliards en circulation dans le monde actuel. Ils pourraient être plus de 50 milliards d'ici 2020.

Pour connecter les objets, les technologies actuelles peuvent être utilisées (`RFID`, `Wi-Fi`, `GSM`, `Bluetooth`, `Z-Wave`, `ZigBee`, ...) mais de nouveaux réseaux se développent au niveau mondial. Ils doivent permettre d'envoyer (mais aussi recevoir dans certains cas) **des très petits messages sur des longues portées** sans passer par des systèmes coûteux de réseaux mobile et **en consommant peu d'énergie**.

> Au cours de ce TP, vous allez mettre en œuvre un objet connecté à l'internet des objets par le réseau LoRa de `The Things Network` et réaliser une application Qt pour interagir avec lui.

2. Objectif de l’activité

- Réaliser un "node" ou objet connecté pour superviser des actions logique et analogique.
- Intégrer un "node" ou objet connecté dans le "cloud" TTN (The Thing Network)
- Réaliser une application Qt cliente pour interagir avec l’objet.

3. The Things Network

Visiter le site web officiel The Things Network : https://www.thethingsnetwork.org/

3.1. Présentation

`The Things Network` est un réseau communautaire fonctionnant avec la technologie LoRa conforme au standard LoRaWAN. Cette technologie permet à des objets de transmettre de petits volumes d’informations en consommant très peu d’énergie.

Figure 1. Principe de l’Internet des Objets

La communauté déploie des passerelles connectées au réseau Internet. Les passerelles sont utilisables par tout le monde, permettant à vos objets de continuer à émettre des données vers vos applications, même s’ils sont à des milliers de kilomètres de chez vous, en utilisant les passerelles à proximité. De la même façon, vos passerelles sont susceptibles de recevoir des données d’objets ne vous appartenant pas. Votre passerelle transmet les informations vers l’infrastructure centrale sur réseau qui se charge de les acheminer vers les applications qui les utilisent.

The Things Network développe une passerelle économique pour supporter le déploiement communautaire du réseau, mais il est possible d'utiliser des passerelles d'autres fabricants sur ce même réseau.

Figure 2. https://www.thethingsnetwork.org/docs/gateways/gateway/

Plus le nombre de passerelles déployées est grand meilleur est la couverture. Aujourd'hui les Pays-Bas ont plusieurs zones à forte densités permettant un déploiement efficace d'applications. D'autres réseaux d'opérateurs reconnus, tel Orange, Bouygue, KPN et d'autres dans différents pays sont disponibles mais leur utilisation est soumise à abonnement payant, avec des garanties associées, qui peuvent ne pas être nécessaire dans le cadre de votre usage.

The Things Network vous permet de profiter d'un service similaire à coût nul.

Figure 3. Implantation des passerelle LoRa TTN en janvier 2020

3.2. Installation d’une passerelle

ATTENTION : Cette partie n’est pas à réaliser dans le cadre du TP. La passerelle que vous utiliserez est déjà installée et configurée. Les login et mot de passe à utiliser pour s’y connecter vous seront communiqués par le formateur.

Pour installer la passerelle `The Things Network`, vous devez disposer d’un ordinateur muni d’une carte wifi. En effet, la seconde étape consiste à se connecter à un réseau wifi fournit par la passerelle.

L’installation se fait en 4 étapes :

- Enregistrement
- Connexion
- Configuration
- Premier message

Pour procéder à l’enregistrement de la passerelle sur les serveurs **TheThingsNetwork.org** (TTN), connectez-vous à l’adresse suivante : https://activate.thethingsnetwork.org/ Vous trouverez à l’adresse suivante un tutoriel vidéo pour vous guider dans les différentes étapes :

Suivre le tutoriel vidéo

4. Administration de la passerelle et des applications

Elle se fait à partir de la console TTN accessible ici : https://console.thethingsnetwork.org/

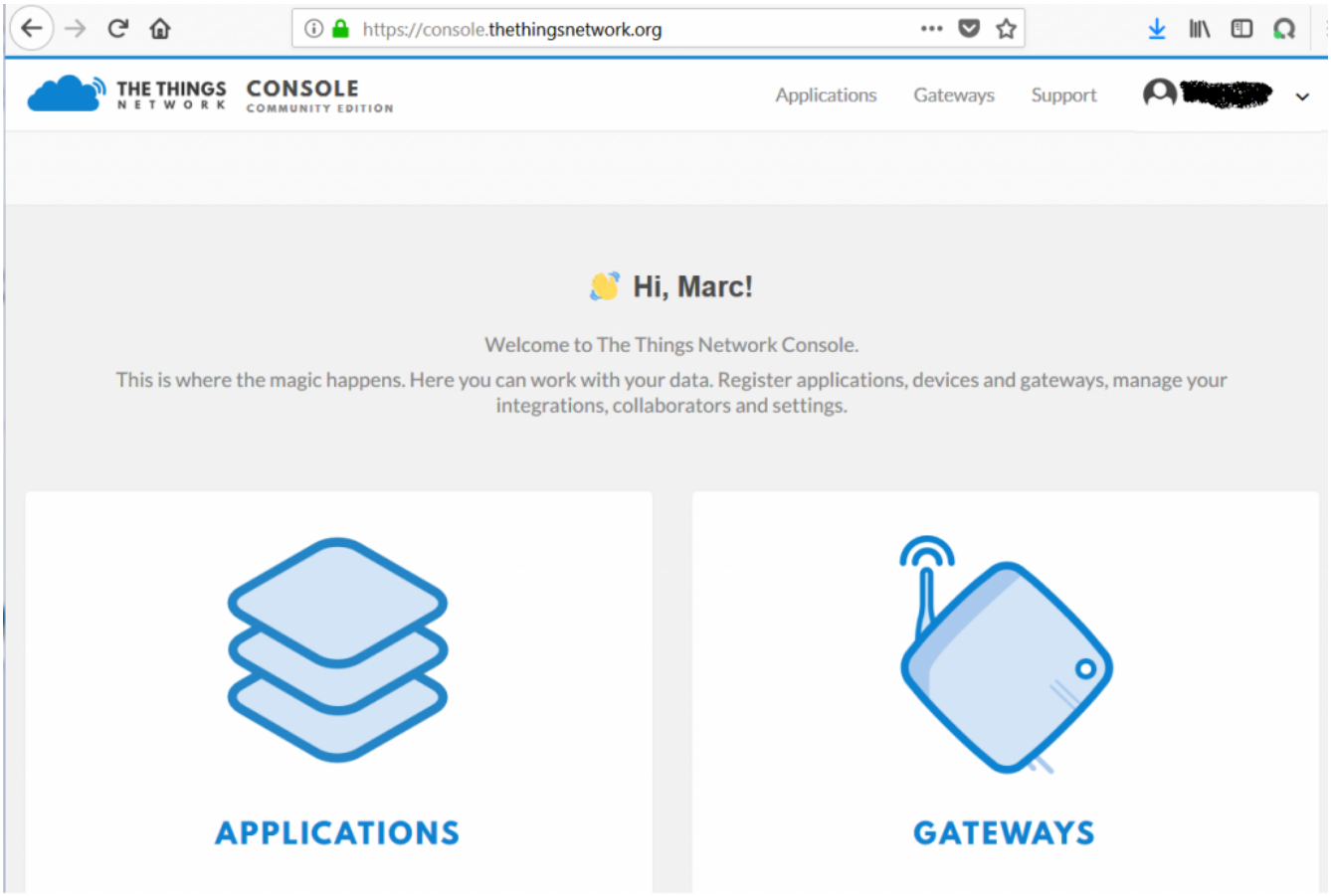


Figure 4. Console The Things Network

4.1. Administration

ATTENTION : NE PAS MODIFIER LA CONFIGURATION DE LA PASSERELLE.

Pour modifier ou compléter la configuration de la passerelle, cliquez sur GATEWAY. Vous pourrez ainsi la géolocaliser, indiquer si elle est placée à l'intérieur ou à l'extérieur, de quel type d'antenne elle est dotée, qui l'administre, ...

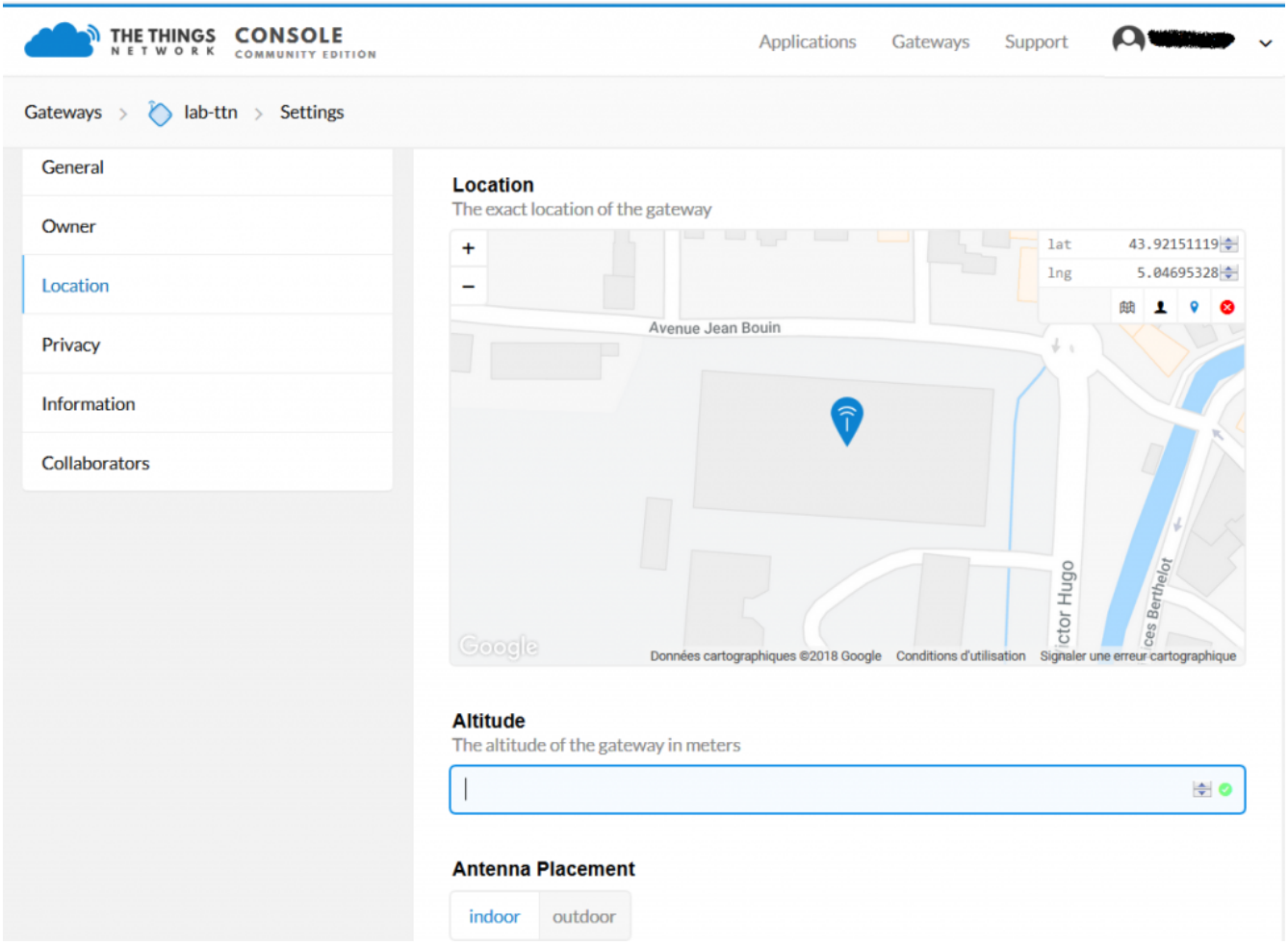


Figure 5. Gestion de la passerelle TTN

4.2. Ajout d'une application

- Documentation : https://www.thethingsnetwork.org/docs/applications/
- Tutoriel video : https://www.youtube.com/watch?v=JrNjY-pGuno

Pour ajouter une application, à partir de la console, cliquez sur `APPLICATIONS`.

Par **Applications**, il faut entendre tout ce que vos objets communiquent sur Internet. On peut également voir une application comme une collection d'objets (`Devices`).

Pour pouvoir enregistrer un objet connecté sur la passerelle, il faut nécessairement disposer d'une application dans laquelle le ranger.

Prenons l'exemple de capteurs et d'actionneurs qui contribuent à la gestion d'un bâtiment qui s'appellerait `Le Machin` dans la ville de `Truc`. On pourrait créer une application nommée `le-machin-a-truc` dans laquelle on déclarera les capteurs `capteur-type-x` ou `type=temperature, humidite, presence, luminosite, ⋯` et `x` un numéro d'identification, idem pour les actionneurs.

Si vous ajouter votre première application, vous pouvez cliquer sur `Get started by adding one`, sinon cliquez sur `+ add application`.

- Complétez dans le formulaire les champs `Application ID` et `Description`. Laissons `The Things Network` décider de l'attribution d'un identifiant `Application EUI` (`Extended Unique Identifier`).
- Le serveur avec lequel les échanges de données se feront sera ici `ttn-handler-eu`.
- Cliquez sur le bouton `Add application` en bas de la page.

Nous pouvons maintenant ajouter des objets connectés à cette application.

Vous pouvez tous utiliser la même application mais la modification du format des données impacte l'ensemble des objets. Il est donc préférable que vous ayez votre propre application, même si elle ne possède qu'un seul objet.

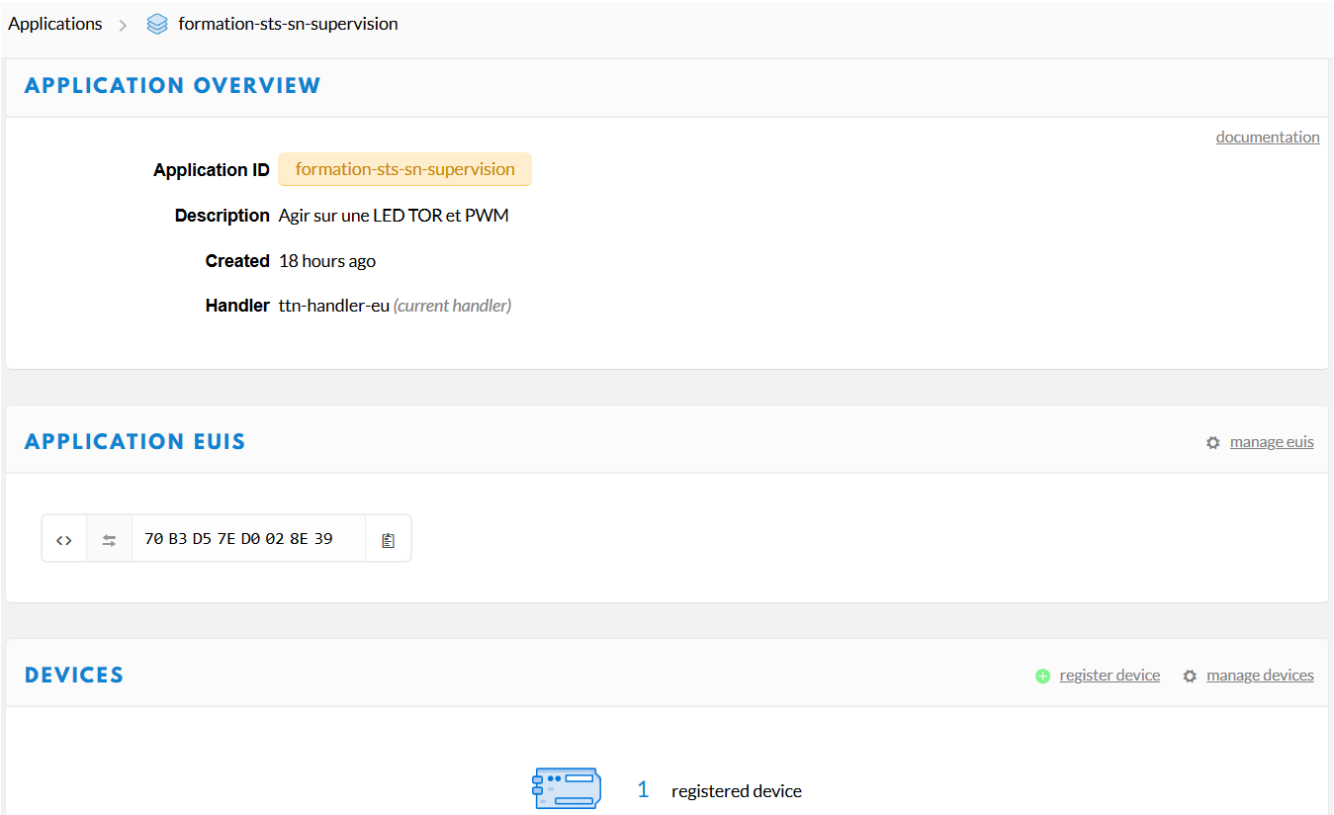


Figure 6. Gestion des applications TTN

5. Ajout d'un objet connecté

- Documentation : https://www.thethingsnetwork.org/docs/devices/registration.html
- Tutoriel vidéo : https://www.youtube.com/watch?v=28Fh5OF8ev0

Nous allons ajouter un objet connecté utilisant une carte `MKR WAN 1300` construite à partir d'une carte `Arduino MKR Zero` (format) et qui intègre une puce LoRa de chez **Murata** et on lui connectera une LED sur une broche PWM.

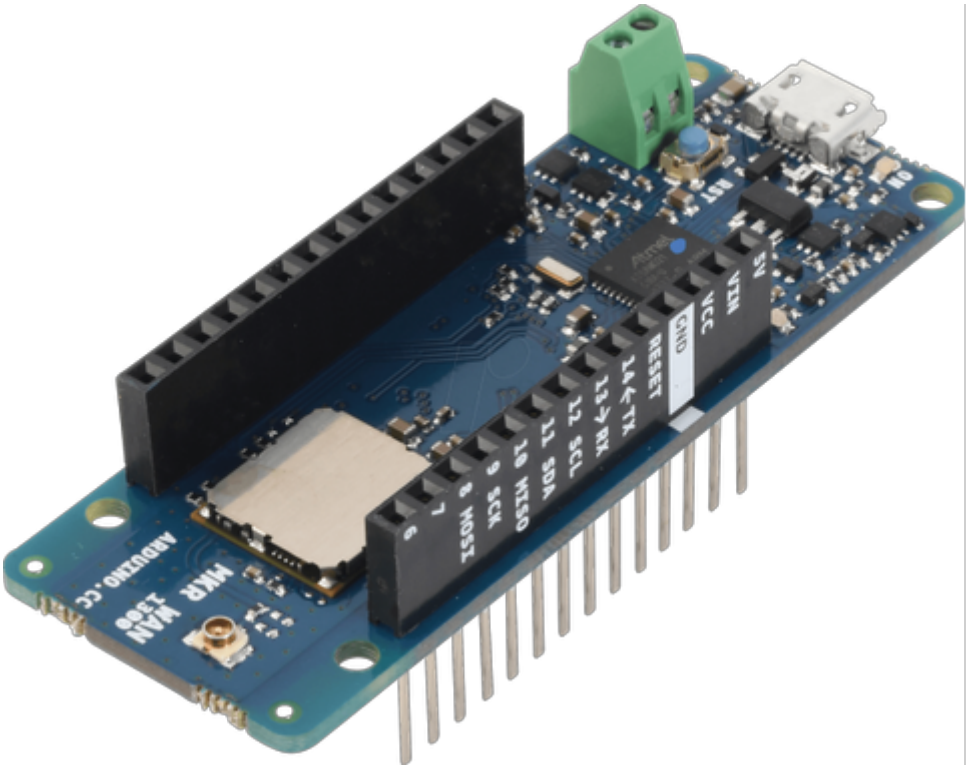


Figure 7. Carte MKR WAN 1300

Arduino MKR WAN 1300 (LoRa connectivity)

5.1. Enregistrement d'un nouvel objet connecté

- Cliquez sur `register device` pour ajouter un objet connecté à l'application.

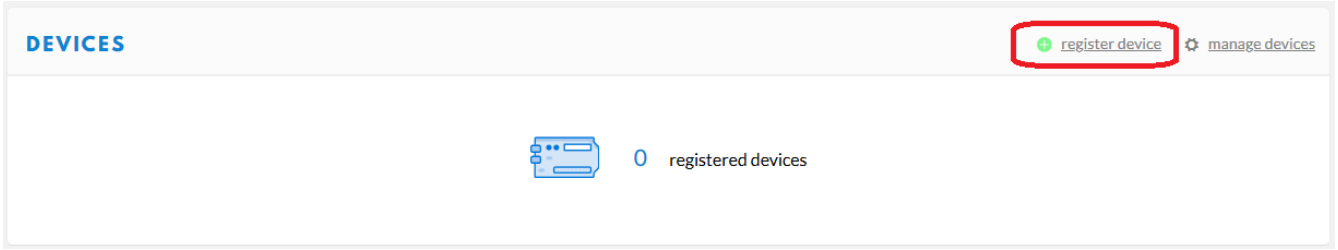


Figure 8. Enregistrer un objet connecté

A ce stade, le seul élément nécessaire à configurer est le `Device ID` : `actionneur-x` (`x =1..12`). Il nous faut également renseigner le `Device EUI` qui est un identifiant propre à la puce LoRa utilisée. On peut toutefois en indiquer un manuellement mais dans notre cas nous allons lire le `DevEUI` de la puce en utilisant un programme d'exemple fourni avec la librairie `TheThingsNetwork` d'Arduino.

Pour l'instant, laissons `The Things Network` le générer automatiquement :

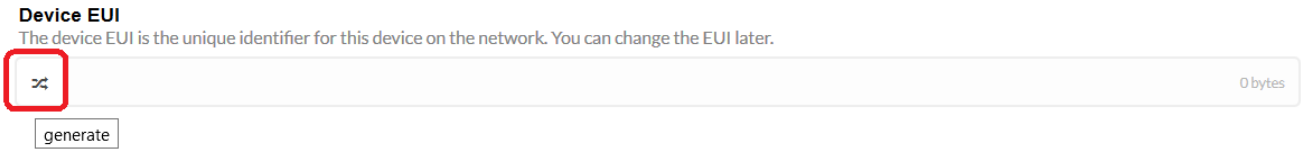


Figure 9. Générer un DevEUI automatiquement

- Cliquez sur le bouton `Register`.

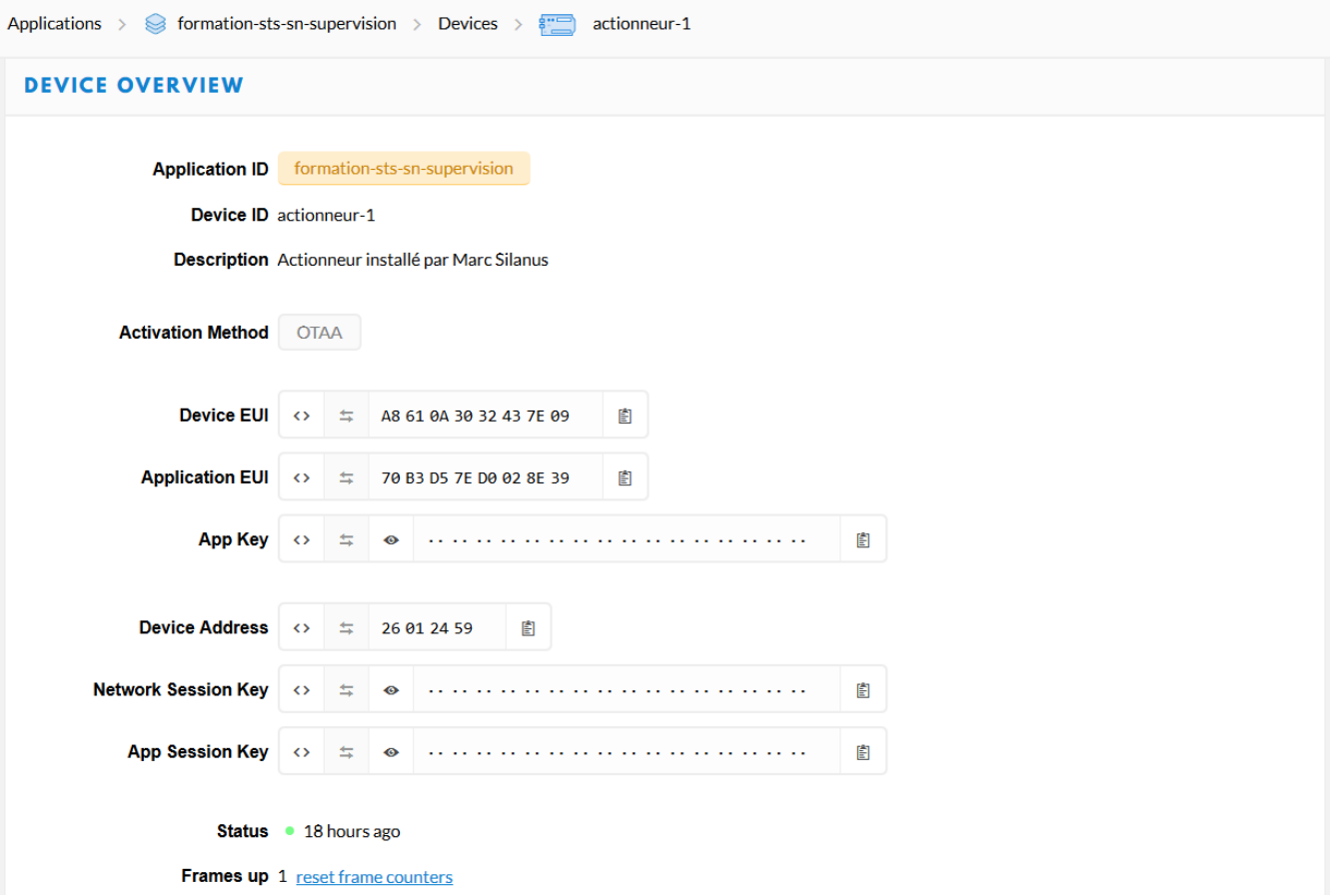


Figure 10. Vue d'ensemble de la configuration d'un objet connecté

L'enregistrement à générer un `Device EUI` aléatoire. Nous allons par la suite le remplacer par l'authentique.

Le champs `Activation Method` définie la manière dont les équipements vont obtenir les clés de session nécessaire à l'établissement de la communication entre eux. Deux types de procédures d'activation sont disponibles :

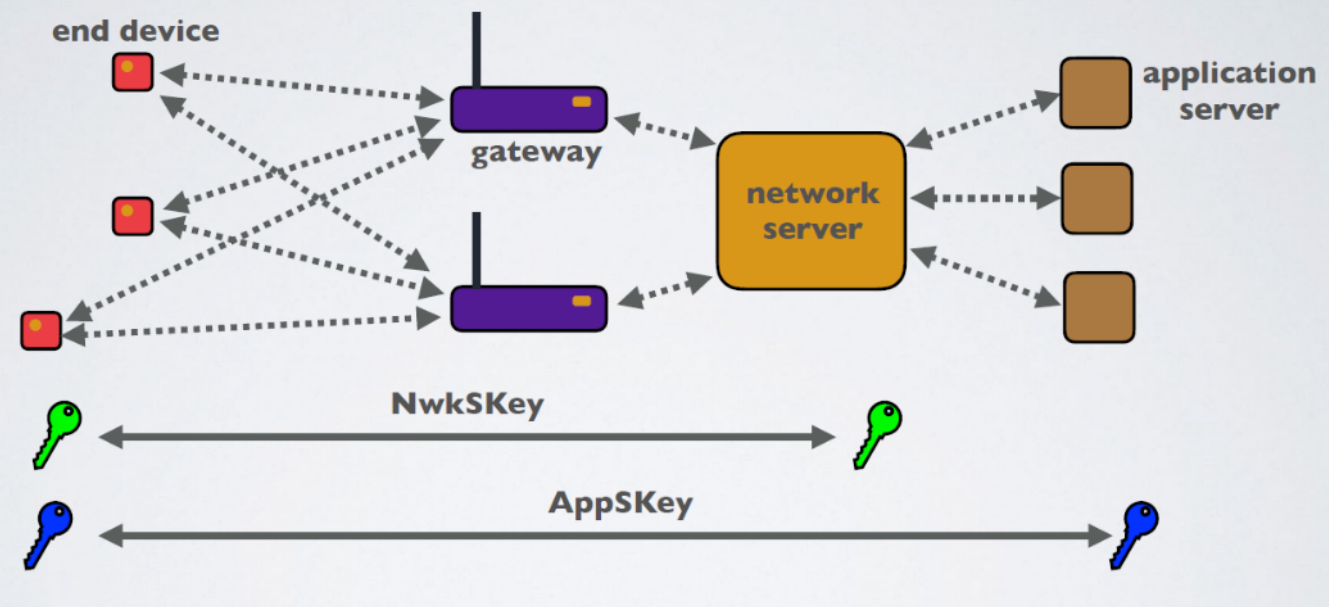
- **Over-The-Air Activation** (`OTAA`)
- **Activation By Personalization** (`ABP`).

Nous choisirons la méthode d'activation `OTAA`.

5.2. Rappel : Méthodes d'activation

L'activation a pour but d'établir une communication sécurisée entre les équipements. Elle est réalisée à partir d'une paire de clés `NwkSKey` pour chiffrer/déchiffrer les données entre le node et le serveur réseau et `AppSKey` pour chiffrer/déchiffrer les données entre le node et le serveur d'applications.

Sécurisation des échanges LoRaWAN



Méthode ABP

Pour la seconde méthode, **ABP**, les clés de session `NwkSKey` et `AppSKey` ainsi que l'adresse de l'équipement (`DevAddr`) sont directement inscrits dans l'équipement LoRaWAN. Ainsi, l'équipement n'a plus besoin d'envoyer de requête avant de communiquer sur le réseau.

L'utilisation de cette méthode implique que les équipements communiquent avec un réseau spécifique car les clés de session sont connues par avance. Et contrairement à la méthode **OTAA**, les clés de session sont statiques.

En résumé, la méthode **OTAA** est plus complexe à implémenter que la méthode **ABP** mais offre un niveau de sécurité supérieur.

Dans le cas d'un prototypage ou pour une utilisation sur un réseau connu, la méthode **ABP** suffit largement. Par contre, lorsqu'un déploiement à plus grande échelle est envisagé, il est conseillé d'utiliser la méthode **OTAA**, plus sécurisée et plus agile.

Méthode OTAA

Pour activer un équipement sur le réseau par la méthode `OTAA`, l'équipement doit transmettre au réseau une demande d'accès : `join request`. Pour ce faire, celui-ci doit être en possession de trois paramètres :

- `DevEUI`, identifiant unique (de type EUI64) de l'équipement (fourni par l'équipementier).
- `AppEUI`, identifiant du fournisseur de l'application (EUI 64).
- `AppKey`, clé AES 128 déterminée par le fournisseur de l'application.

L'équipement envoie, à travers le réseau, la requête `join request`, contenant `DevEUI`, `AppEUI` ainsi qu'un `MIC` (`Message Integriry Code`) calculé via la clé `AppKey`. Cette requête est transmise au serveur d'enregistrement qui vérifie le MIC via la clé AppKey (qui lui a été communiquée au préalable).

Si l'équipement est autorisé par le serveur d'enregistrement, la requête `join accept` est transmise en réponse à l'équipement. Cette réponse contient des données à partir desquelles l'équipement va pouvoir calculer les **clés de session** (**réseau** : `NwkSKey` et **applicative** : `AppSKey`).

Parmi les données contenues dans cette réponse, se trouve également l'adresse **Device Adress** (`DevAddr`) sur 32 bits – qu'utilisera l'équipement pour communiquer sur le réseau.

A chaque nouvelle session, les clés de session sont renouvelées.

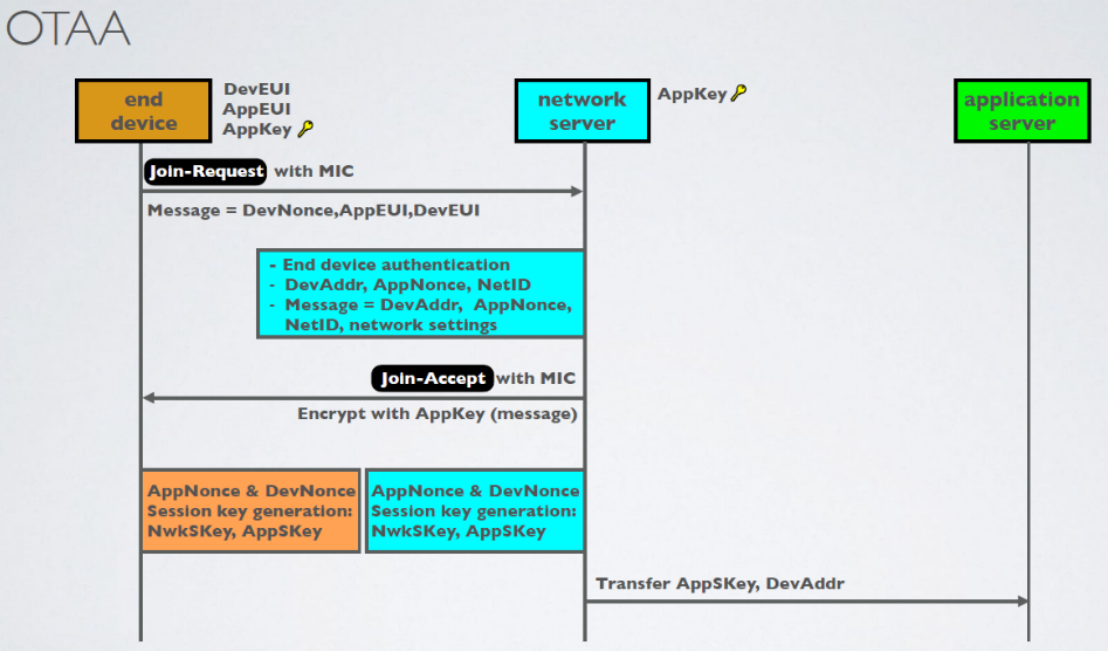


Figure 11. Principe de l' activation OTAA

5.3. Configuration du Device Extended Unique Identifier (DevEUI)

Pour obtenir le DevEUI, commençons par installer le gestionnaire de cartes MKR et la librairie Arduino MKRWAN dans l'EDI Arduino :

- Dans l'environnement de développement d'Arduino, cliquez sur Outils→Type de carte→Gestionnaire de carte.
- Dans la barre de recherche, saisissez MKR 1300
- Procédez à l'installation du gestionnaire de cartes.

Figure 12. Installation de la carte MKR WAN 1300

- Dans l'environnement de développement d'Arduino, cliquez sur Croquis→Inclure une bibliothèque→Gérer les bibliothèques.
- Dans la barre de recherche, saisissez MKR 1300
- Procédez à l'installation de la librairie

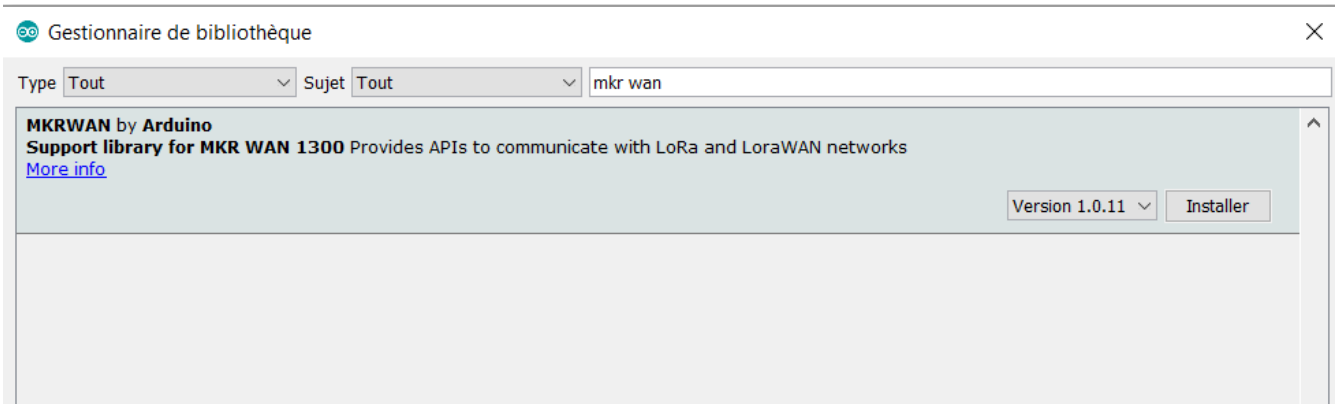


Figure 13. Installation de la librairie MKRWAN

Nous disposons maintenant de la librairie et de ses programmes d'exemples. Nous allons ouvrir le programme FirstConfiguration.ino :

Fichier→Exemples→MKRWAN→FirstConfiguration

- Assurez vous que le plan de fréquence est bien celui pour l'Europe.
- Connectez la carte MKR WAN 1300 au PC.

- Indiquez à l'IDE que vous utilisez une carte de type `MKR WAN 1300`.
- Compilez et téléversez.
- Ouvrez le terminal pour lire les infos :

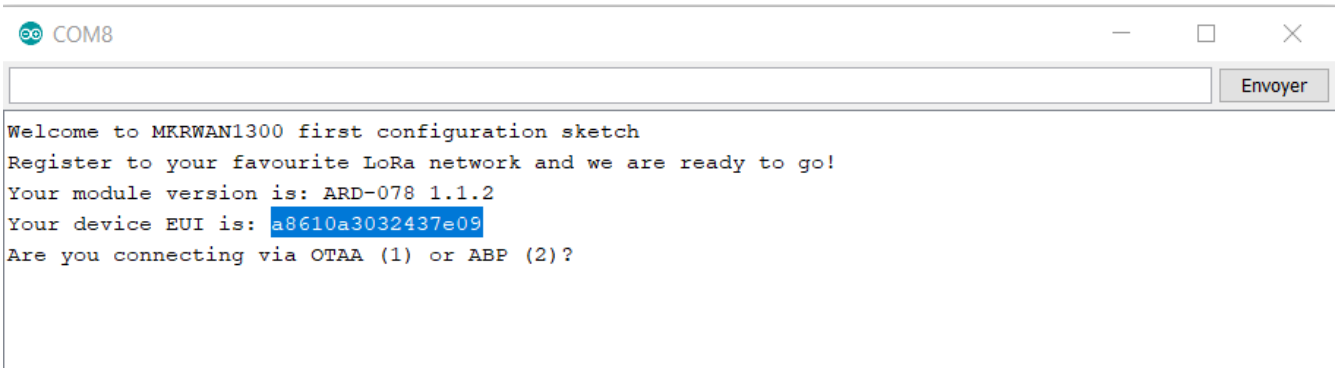


Figure 14. Lecture du DevEUI de la puce LoRa

- Configurez dans la console TTN le `DevEUI` :
  - Dans la page de l'objet connecté, cliquez sur `settings`
  - Remplacez le `DevEUI` précédemment généré aléatoirement par celui de la puce LoRa

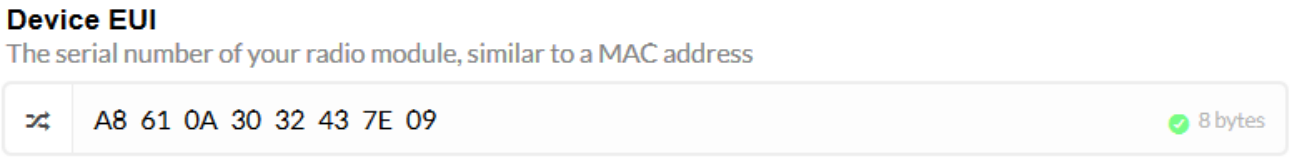


Figure 15. Configurer le DevEUI d'un objet dans la console TTN

- Ajoutez une description qui contient votre nom à l'objet (pratique pour la gestion des objets enregistrés).
- Cliquez sur `Save` pour enregistrer l'objet.

`The Things Network` a généré les clés `AppKey` et `AppEUI` nécessaire à l'établissement de la liaison entre l'objet et le serveur TTN :

Figure 16. AppEUI et AppKey générées par The Things Network pour l'objet

5.4. Envoyer et recevoir

Dans l'IDE d'Arduino :

- Ouvrez le programme d'exemple **LoraSendAndReceive** : `Fichier→Exemples→MKRWAN→LoraSendAndReceive`
- Assurez-vous qu'on utilise le bon plan de fréquence et configurez `AppEUI` et `AppKey` conformément aux informations contenues dans la déclaration du `device` dans la console TTN.

Figure 17. Renseigner les AppEUI et AppKey dans l'objet

- La boucle principale du programme lit une chaine de caractères entrée depuis le terminal série et attend un retour à la ligne (`'\n'`) avant de le transmettre sur le réseau LoRa.
- Par défaut, les objets connectés développés sur cette carte avec la librairie MKRWAN sont de classe A. On attend donc une seconde après l'envoi de données pour lire les éventuelles données envoyées depuis TTN.

```
void loop() {
  Serial.println();
  Serial.println("Enter a message to send to network");
  Serial.println("(make sure that end-of-line 'NL' is enabled)");

  while (!Serial.available());
  String msg = Serial.readStringUntil('\n');

  Serial.println();
  Serial.print("Sending: " + msg + " - ");
  for (unsigned int i = 0; i < msg.length(); i++) {
    Serial.print(msg[i] >> 4, HEX);
    Serial.print(msg[i] & 0xF, HEX);
    Serial.print(" ");
  }
  Serial.println();

  int err;
  modem.beginPacket();
  modem.print(msg);
  err = modem.endPacket(true);
  if (err > 0) {
    Serial.println("Message sent correctly!");
  } else {
    Serial.println("Error sending message :(");
    Serial.println("(you may send a limited amount of messages per minute, depending
on the signal strength");
    Serial.println("it may vary from 1 message every couple of seconds to 1 message
every minute)");
  }
  delay(1000);
  if (!modem.available()) {
    Serial.println("No downlink message received at this time.");
    return;
  }
  char rcv[64];
  int i = 0;
  while (modem.available()) {
    rcv[i++] = (char)modem.read();
  }
  Serial.print("Received: ");
  for (unsigned int j = 0; j < i; j++) {
    Serial.print(rcv[j] >> 4, HEX);
    Serial.print(rcv[j] & 0xF, HEX);
    Serial.print(" ");
  }
  Serial.println();
}
```

Nous allons adapter le programme pour qu'il transmette l'état de la `LED_BUILTIN` conncetée sur cette carte à la broche `D6` et qu'il écoute les ordres de commande ce cette LED :

- Configurez la broche `D6` en sortie

```
void setup(){
  ...
  pinMode(LED_BUILTIN,OUTPUT);
}
```

Nous n'avons besoin que d'un bit pour commander la LED, nous n'utiliserons donc qu'un octet pour transmettre un ordre depuis TTN et nous transmettrons directement directement son état lu sur la carte :

```
void loop() {
  // Lecture de l'état de la LED
  String led_state = String(digitalRead(LED_BUILTIN));
  Serial.println("LED_BUILTIN State : " + led_state);

  // Trnsmission de l'état de la LED
  int err;
  modem.beginPacket();
  modem.print(led_state);
  err = modem.endPacket(true);
  if (err > 0) {
    Serial.println("Message sent correctly!");
  } else {
    Serial.println("Error sending message :(");
    Serial.println("(you may send a limited amount of messages per minute, depending
on the signal strength");
    Serial.println("it may vary from 1 message every couple of seconds to 1 message
every minute)");


  // Attente pour une éventuelle réponse
  delay(1000);
  if (!modem.available()) {
    Serial.println("No downlink message received at this time.");
    return;
  }

  // Lecture des ordres envoyés depuis TTN
  char rcv[64];
  int i = 0;
  while (modem.available()) {
    rcv[i++] = (char)modem.read();
  }
  Serial.print("Received: ");
  for (unsigned int j = 0; j < i; j++) {
    Serial.print(rcv[j] >> 4, HEX);
    Serial.print(rcv[j] & 0xF, HEX);
    Serial.print(" ");
  }
  // Seul le premier caractère reçu nous intéresse
  digitalWrite(LED_BUILTIN,rcv[0]);

  Serial.println();
}
// Attendre avant de recommencer !
delay(60000);
```

- Compilez, téléversez et observez le terminal série :

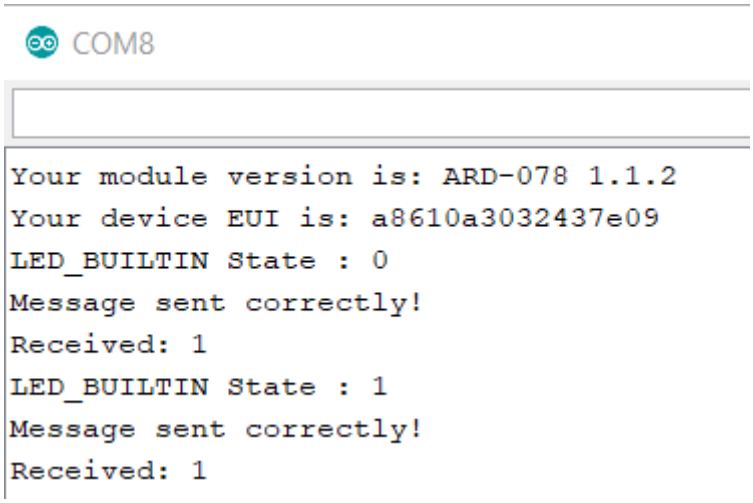


Figure 18. Trace debug du programme dans le terminal

- Observez les données dans la console TTN :

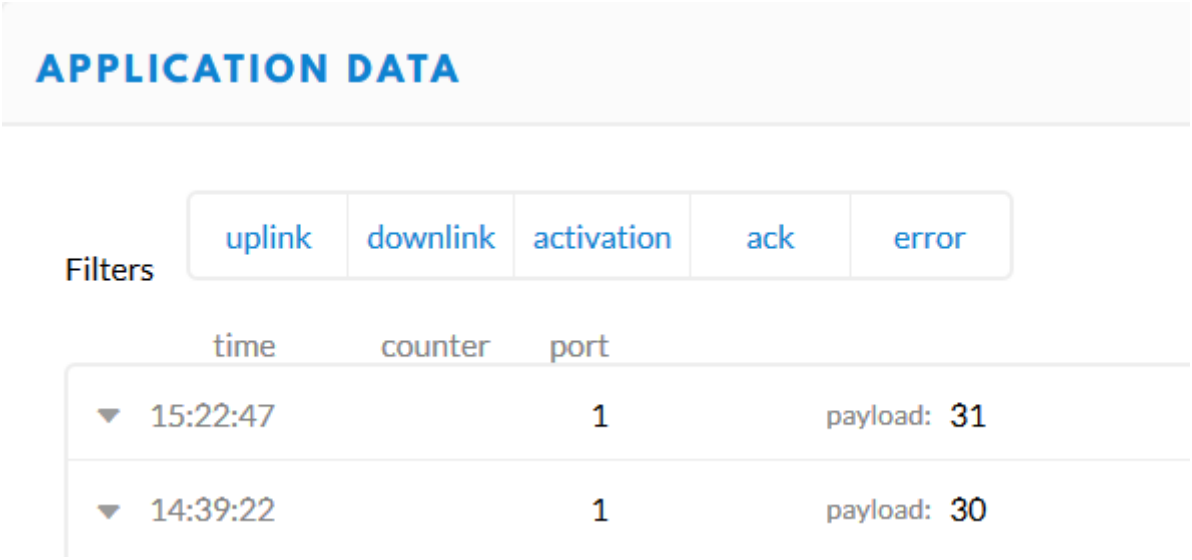


Figure 19. Observation des données arrivées sur les serveurs TTN

- Conformément à la programmation de l'objet, on lit **1 octet** dont la valeur est `0X30 ('0')` ou `0X31 ('1')` qui reflète l'état de la LED.
- Proposez une fonction de décodage pour rendre les données aisément lisible : Cliquez sur le menu `Devices` puis sur le bouton `Payload formats` et modifiez l'exemple de fonction `Decoder(bytes, port)` :

```
function Decoder(bytes) {
  // Decode an uplink message from a buffer
  // (array) of bytes to an object of fields.
  var decoded = {};

  decoded.led = bytes[0]-48;

  return decoded;
}
```

- Observez les données dans la console TTN.

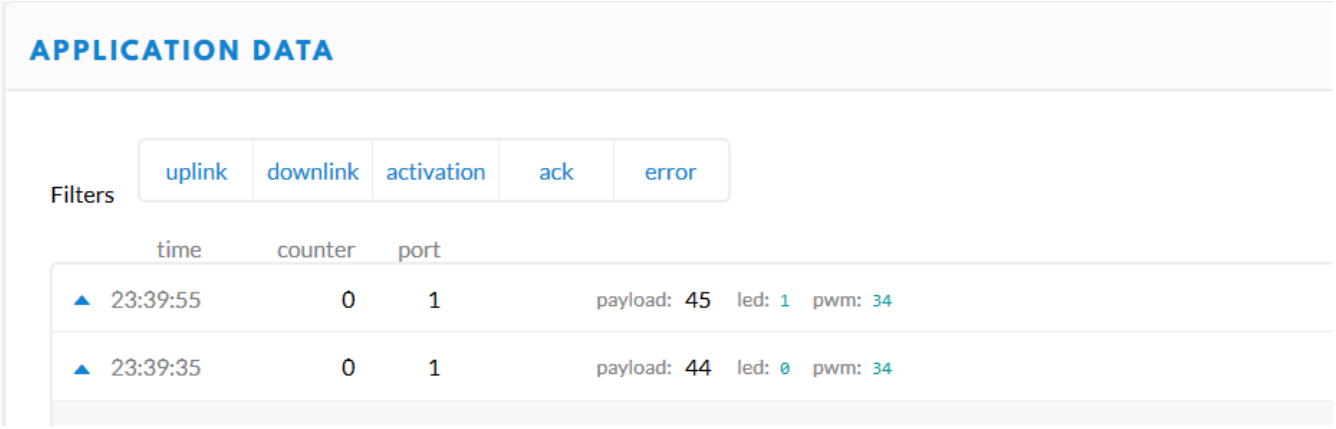


Figure 20. Observation des données après formatage

6. Serveur d'application

`The Things Network` diffuse les données des objets connectés via le protocole **MQTT** avec les éléments de configuration suivant :

- Host: `<Region>.thethings.network`, where `<Region>` is last part of the handler you registered your application to, e.g. `eu`.
- Port: `1883`
- Username: `Application ID`
- Password: `Application Access Key`
- Topic: `<AppID>/devices/<DevID>/up`

Références de l'API MQTT de TTN

De nombreuses solutions sont alors possibles pour la mise en place d'un serveur d'application :

- Application MQTT client (HiveMQ, MQTT cli, mqtt-spy, ...)

- Node Red : Node-RED est un outil puissant pour construire des applications de l'Internet des Objets en mettant l'accent sur la simplification de la programmation qui se fait grâce à des blocs de code prédéfinis, appelés «**nodes**» pour effectuer des tâches. Il utilise une approche de programmation visuelle qui permet aux développeurs de connecter les blocs de code ensemble. Les noeuds connectés, généralement une combinaison de noeuds d'entrée, de noeuds de traitement et de noeuds de sortie, lorsqu'ils sont câblés ensemble, constituent un «**flow**». Le dashboard ainsi réalisé est accessible via un navigateur depuis n'importe quel périphérique du réseau.

- Un service d'intégration comme par exemple `Cayenne myDevices`. Cette solution est sans doute la plus rapide à mettre en oeuvre. Elle fait appel à un service extérieur qui se connecte au broker MQTT de `The Things Network` et interprète les données envoyées par l'objet pour établir automatiquement un dashboard. Il faut cependant que les données soient correctement formatées au format `Cayenne LPP` (Cayenne Low Power Payload).

- Une application **Qt** utilisant le module `QtMQTT`.

Je vous propose ici de réaliser une application cliente avec Qt qui accèdera directement aux messages via MQTT au travers de l'utilisation du module `QtMQTT`.

6.1. Installation du module QtMQTT

L'implémentation de MQTT peut également se faire en s'appuyant sur :

- la librairie C/C++ `libmosquitto` : https://github.com/eclipse/mosquitto
- le module `QMQTT` : https://github.com/emqx/qmqtt

L'installation du module `QtMQTT` requiert un interpréteur `PERL`. Celui-ci peut être installé pendant l'installation de `Qt Creator`. Sinon, installer le : https://www.perl.org/get.html

Pour Windows, installer la version **Strawberry**.

Le module `QtMQTT` n'est installé par défaut que dans la version commerciale de Qt spécialisée https://doc.qt.io/QtForAutomation/qtautomation-index.html.

Il est en revanche possible de l'installer manuellement à partir de ses sources :

git://code.qt.io/qt/qtmqtt.git

- Téléchargez les sources :

```
git clone  git://code.qt.io/qt/qtmqtt.git
cd qtmqtt
```

- Double-cliquez sur le fichier `qtmqtt.pro` pour ouvrir le projet dans `Qt Creator`.
- Cliquez sur `Projets` puis ajouter une étape `Make` aux étapes de compilation avec comme argument : `install`

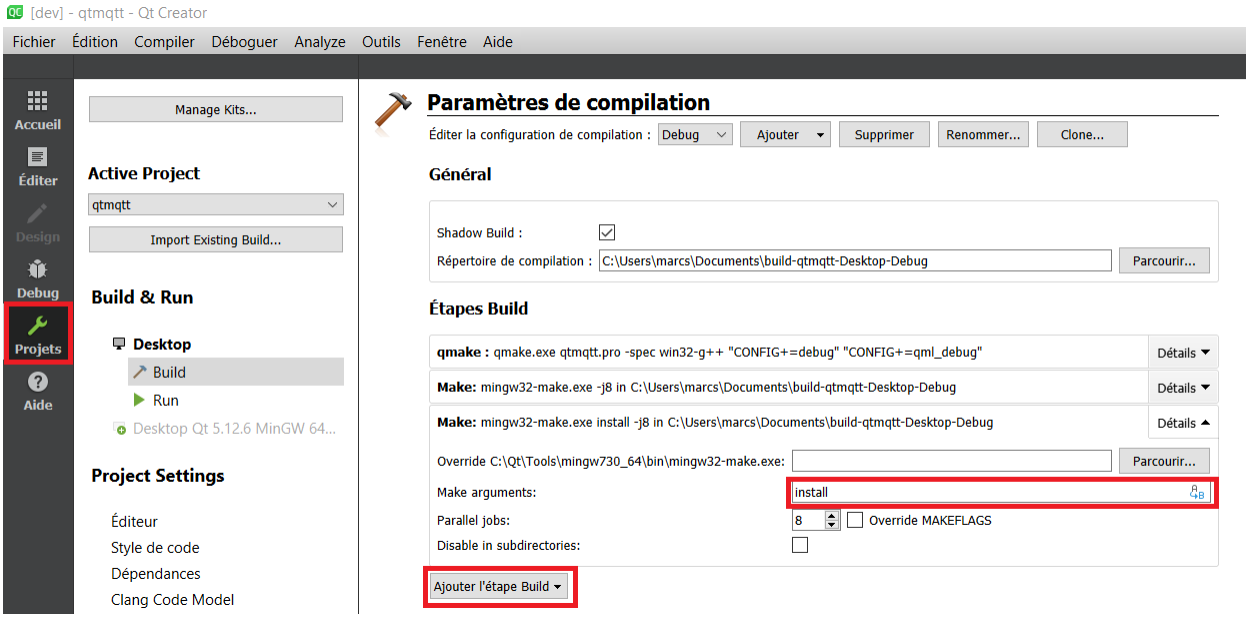


Figure 21. Installation du module depuis Qt

- Compilez le projet.

6.2. Exemple `simpleclient`

Le module dispose de programme d'exemple. Dans le dossier `qtmqtt/exemples/mqtt/simpleclient`, double cliquez sur `simpleclient.pro` pour ouvrir le projet dans Qt.

Fermez le projet en cours (clique droit à la racine du projet et `Fermer le projet`).

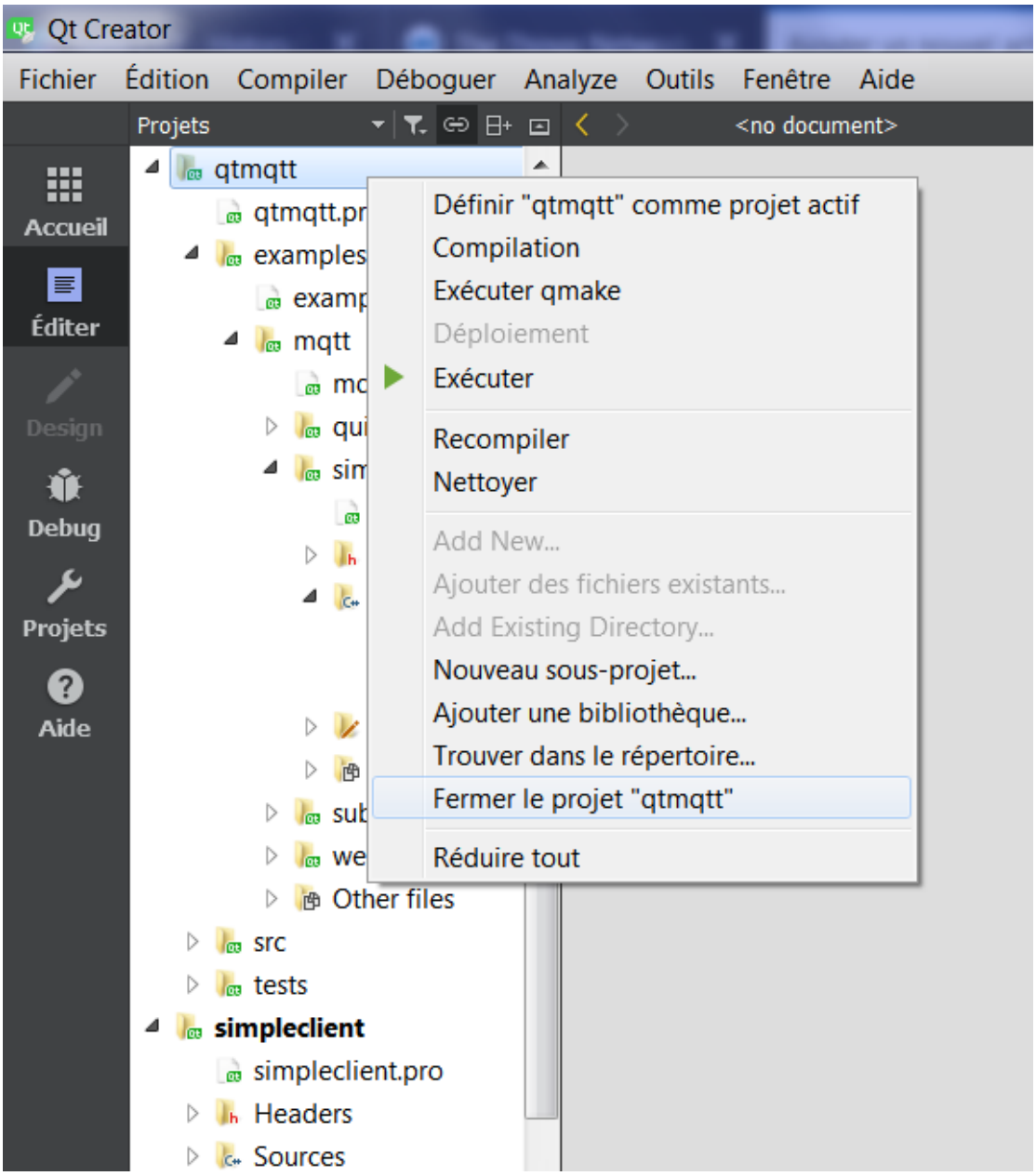


Figure 22. Programme d'exemple simpleclient

- Compilez et exécutez le programme :

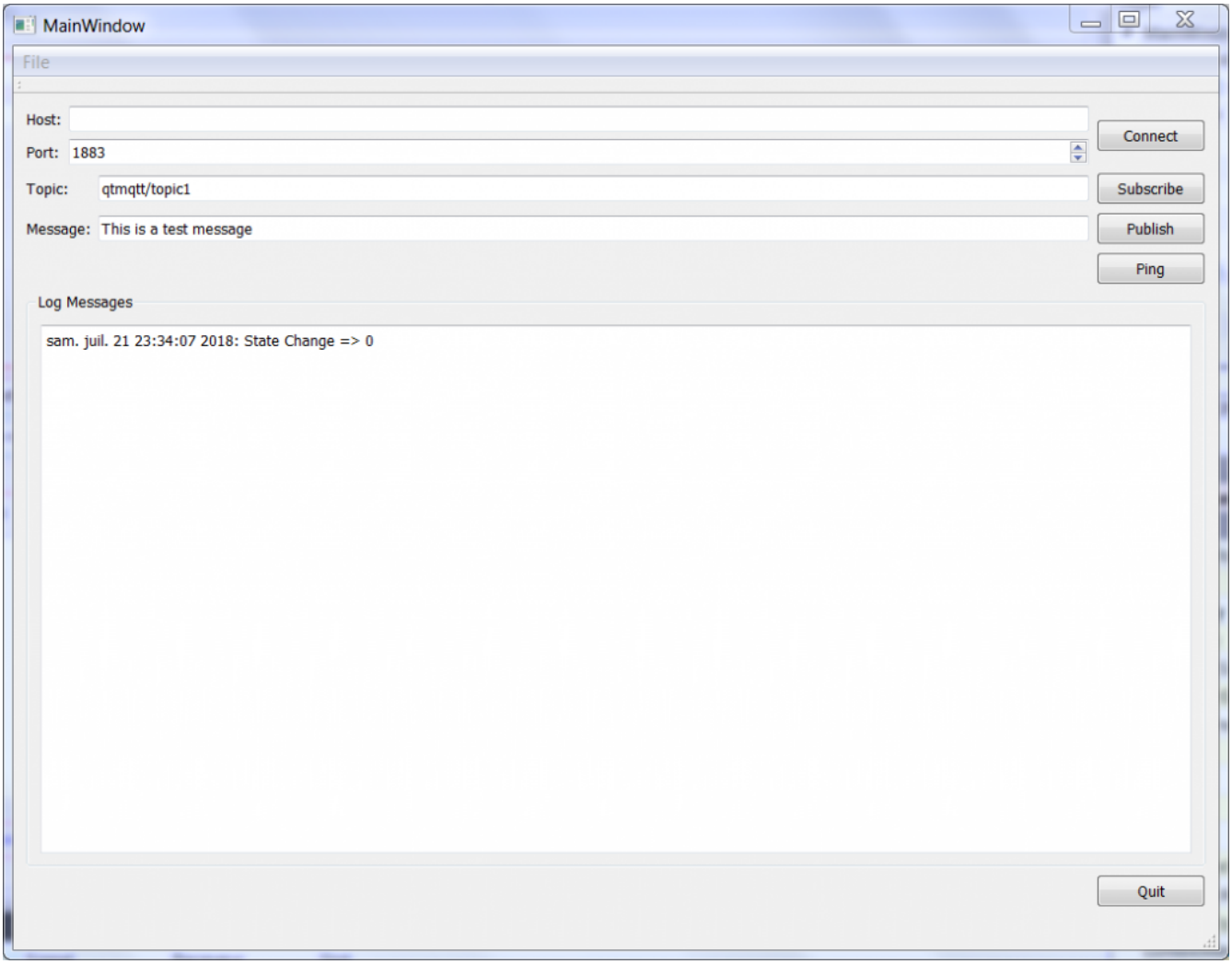


Figure 23. Excution du programme d'exemple simpleclient

- Ouvrez le fichier `mainwindow.cpp` et observez le code du constructeur de la classe `MainWindow`:

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    m_client = new QMqttClient(this);
    m_client->setHostname(ui->lineEditHost->text());
    m_client->setPort(ui->spinBoxPort->value());

    connect(m_client, &QMqttClient::stateChanged, this,
&MainWindow::updateLogStateChange);
    connect(m_client, &QMqttClient::disconnected, this,
&MainWindow::brokerDisconnected);

    connect(m_client, &QMqttClient::messageReceived, this, [this](const QByteArray
&message, const QMqttTopicName &topic) {
        const QString content = QDateTime::currentDateTime().toString()
                    + QLatin1String(" Received Topic: ")
                    + topic.name()
                    + QLatin1String(" Message: ")
                    + message
                    + QLatin1Char('\n');
        ui->editLog->insertPlainText(content);
    });

    connect(m_client, &QMqttClient::pingResponseReceived, this, [this]() {
        const QString content = QDateTime::currentDateTime().toString()
                    + QLatin1String(" PingResponse")
                    + QLatin1Char('\n');
        ui->editLog->insertPlainText(content);
    });

    connect(ui->lineEditHost, &QLineEdit::textChanged, m_client,
&QMqttClient::setHostname);
    connect(ui->spinBoxPort, QOverload<int>::of(&QSpinBox::valueChanged), this,
&MainWindow::setClientPort);
    updateLogStateChange();
}
```

On y trouve un objet `m_client` de la classe `QMqttClient` dont la définition de l'hôte à connecter se fait par l'appel à la méthode `setHostName()` et la définition du port par l'appel à la méthode `setPort()`.

Documentation de la classe QMqttClient.

Dans le code de l'exemple, il n'est pas prévu d'utiliser une connexion sécurisée. Il nous faut donc ajouter le code nécessaire pour indiquer le nom d'utilisateur et le mot de passe, conformément aux paramètres attendus par TTN :

- Host: `<Region>.thethings.network`, where `<Region>` is last part of the handler you registered your application to, e.g. `eu`.
- Port: `1883`
- Username: `Application ID`
- Password: `Application Access Key`
- Topic: `<AppID>/devices/<DevID>/up`

Il nous faut donc utiliser les méthodes :

```
m_client->setUsername(QString);
m_client->setPassword(QString);
```

- Ajustez le code relativement aux indications de la documentation de `The Things Network` et à la configuration de l'application et du `device` déclaré dans le console TTN :

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    m_client = new QMqttClient(this);
    ui->lineEditHost->setText("eu.thethings.network");
    m_client->setHostname(ui->lineEditHost->text());
    m_client->setPort(ui->spinBoxPort->value());

    m_client->setUsername("formation-sts-sn-supervision");
    m_client->setPassword("ttn-account-v2.5h8uByRog3gRkSIHY9-PUDjeFP0GkvmiFXCKGX4p-
20");
   ...
}
```

- Compilez et exécutez le programme, saisissez l'adresse du broker TTN (`Host`), puis cliquez sur le bouton `Connect` :

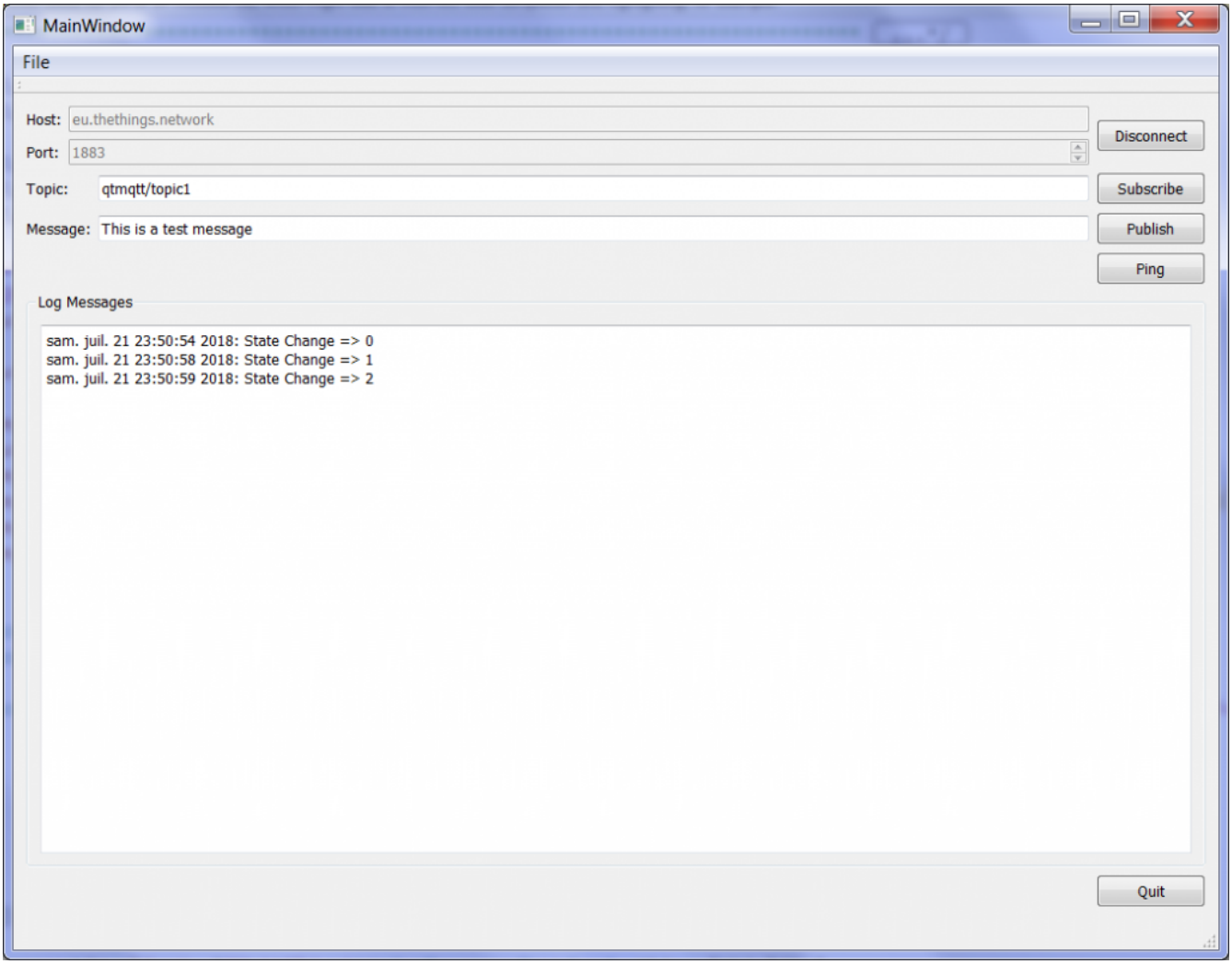


Figure 24. Connexion au broker TTN depuis le programme d'exemple simpleclient

- Constatez le passage à l'état 2 de la connexion entre le client et le broker. D'après la documentation de la méthode `state()`, cela signifie que l'on est connecté :

enum QMqttClient::ClientState

This enum type specifies the states a client can enter.

Constant	Value	Description
`QMqttClient::Disconnected`	0	The client is disconnected from the broker.
`QMqttClient::Connecting`	1	A connection request has been made, but the broker has not approved the connection yet.
`QMqttClient::Connected`	2	The client is connected to the broker.

Figure 25. Signification des valeurs de ClientState

Essayons maintenant de lire les messages. Pour cela, nous devons souscrire au Topic `<AppID>/devices/<DevID>/up`, soit pour moi :

`formation-sts-sn-supervision/devices/actionneur-1/up`

- Renseignez le champs `Topic` du formulaire et cliquez sur le bouton `Subscribe` :

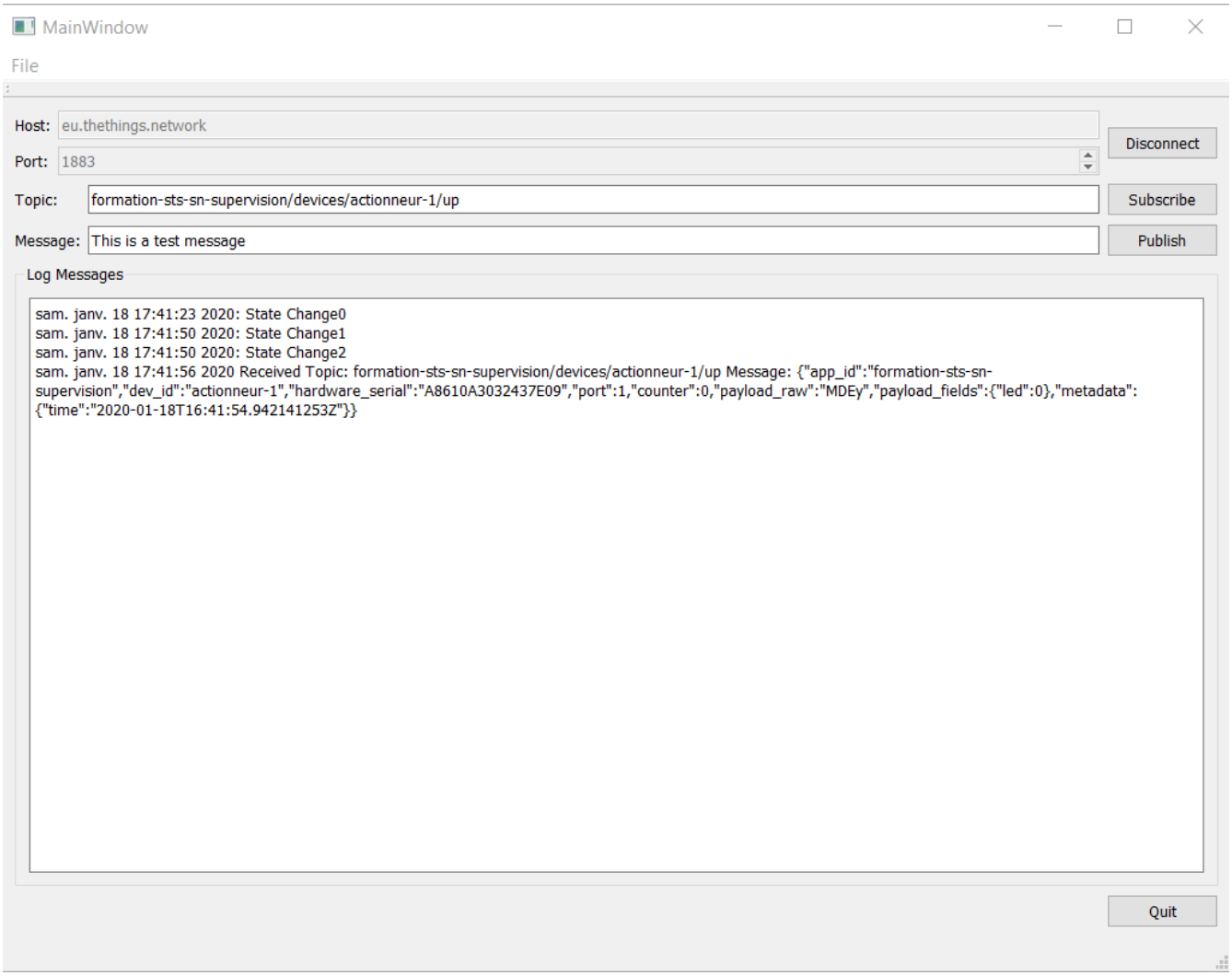


Figure 26. Programme d'exemple simpleclient : souscription au topic

On récupère bien le message, il ne restera plus qu'à le traiter !!!

6.3. Notre application

Nous allons réaliser une petite application pour afficher l'état de la LED et la piloter. Nous ajouterons une LED supplémentaire sur la carte Arduino MKR 1300 sur un port PWM afin de gérer également une gradation.

6.3.1. Le protocole

Les besoins pour afficher l'état ou pour piloter sont :

- `LED_BUILTIN` : Allumer ou éteindre soit 1 bits
- `LED_PWM` : 0 à 100% soit 7 bits (0 à 127)

Un octet suffira donc, il n'y aura pas beaucoup de changement à apporter au programme de la carte.

7	6	5	4	3	2	1	0
LED_PWM							LED_BUILTIN

- Modifier le programme Arduino en conséquence

```
byte pwm = 0;
byte led = 0;

...

void loop() {

  led = digitalRead(LED_BUILTIN);
  byte toTTN = pwm << 1 | led;

  String leds_state = String(toTTN);

  Serial.println("LEDS State : " + leds_state);

  int err = 1;
  modem.beginPacket();
  modem.print(leds_state);

  ...

  char rcv[64];
  int i = 0;
  while (modem.available()) {
    rcv[i++] = (char)modem.read();
  }
  Serial.print("Received: ");
  for (unsigned int j = 0; j < i; j++) {
    Serial.print(rcv[j] >> 4, HEX);
    Serial.print(rcv[j] & 0xF, HEX);
    Serial.print(" ");
  }
  pwm = rcv[0] >> 1;
  led = rcv[0] & 1;
  Serial.print("pwm = ");
  Serial.println(pwm);
  Serial.print("led = ");
  Serial.println(led);

  digitalWrite(LED_BUILTIN,led);
  analogWrite(5,map(0,100,0,255));

  Serial.println();
  delay(60000);
```

6.3.2. Arduino ← → TTN

- Modifiez la fonction `Decoder(byte)` dans payload format de TTN pour restituer l'état de la LED et la valeur de la PWM.

```
function Decoder(bytes) {
  // Decode an uplink message from a buffer
  // (array) of bytes to an object of fields.
  var decoded = {};
  var led = bytes[0] & 1;
  var pwm = bytes[0] >> 1;


  decoded.led = led;
  decoded.pwm = pwm;

  return decoded;
}
```

6.3.3. TTN ← → Application Qt

- Créez votre application Qt par exemple :

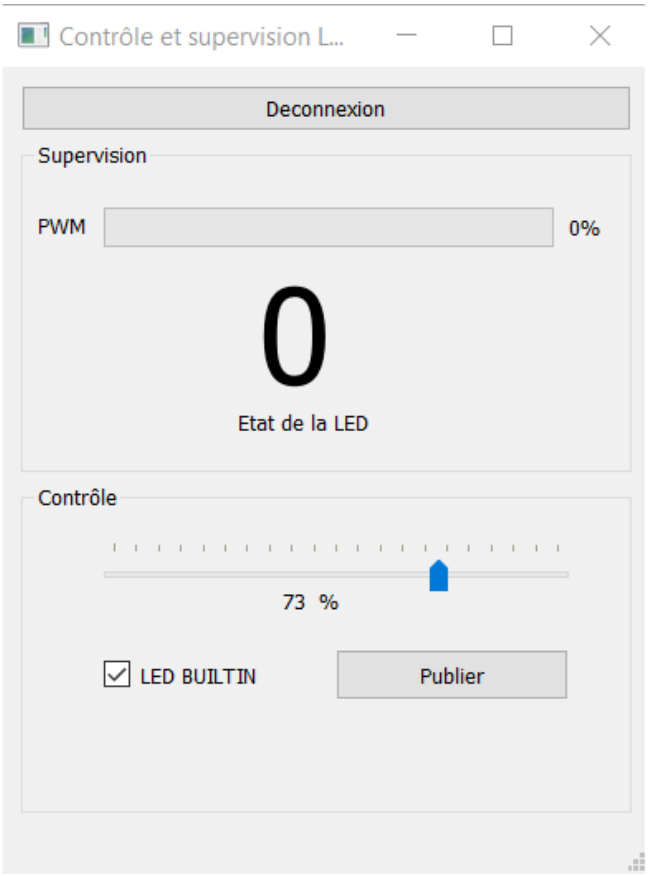


Figure 27. Application en service

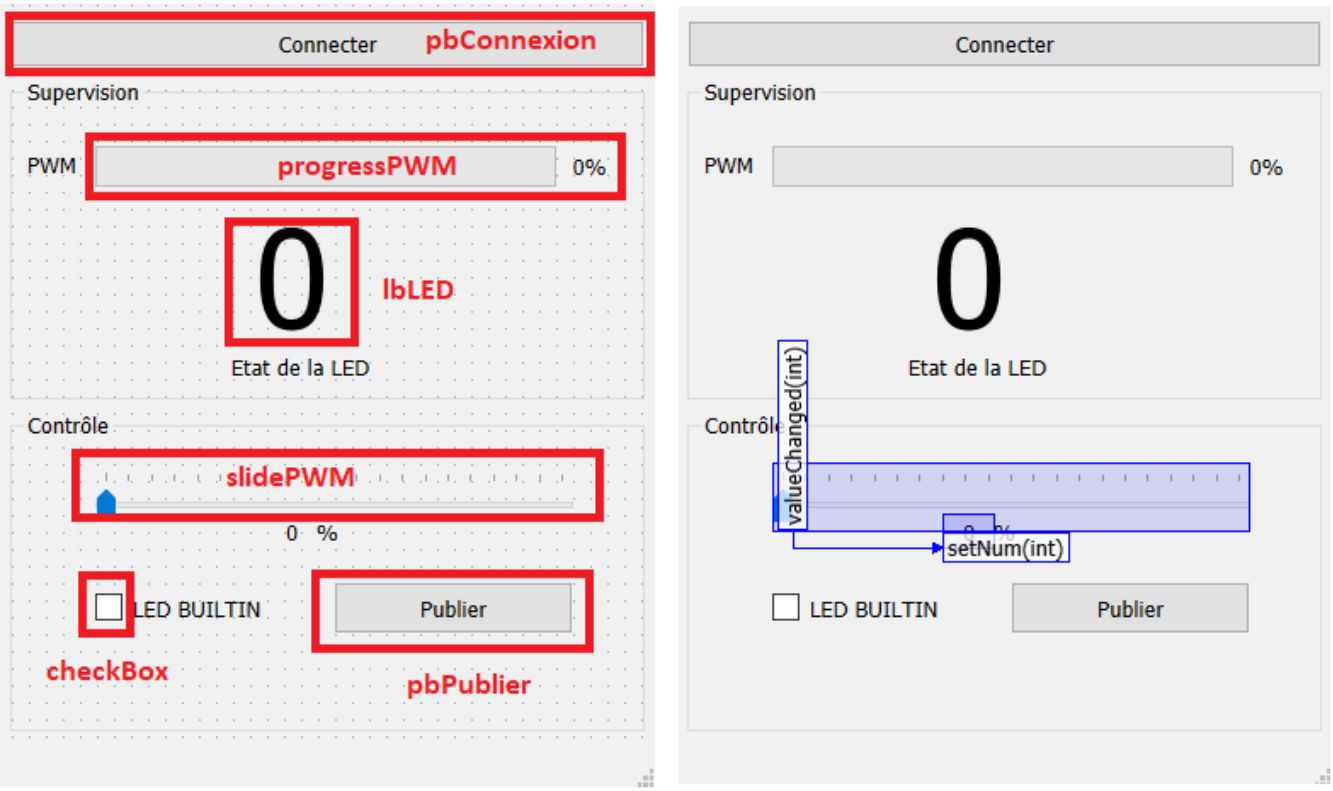


Figure 28. Construction de l'application

6.3.4. Le projet Qt (`.pro`)

- Ajoutez à la définition du projet (`.pro`) la référence au module `QtMQTT`

```
QT       += mqtt
```

6.3.5. Le code

Déclaration d'un objet de la classe `QMqttClient`

- Editez le fichier `mainwindow.h` et ajoutez la librairie `QMqttClient` :

```
#include <QMqttClient>
```

- Déclarez un objet de la classe QMqttClient :

```
private:
    Ui::MainWindow *ui;
    QMqttClient *m_client;
```

La connexion

Dans le fichier `mainwindow.cpp`, instanciez l'objet `m_client` et configurez les paramètres de connexion MQTT :

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    m_client = new QMqttClient(this);
    m_client->setHostname("eu.thethings.network");
    m_client->setPort(1883);
    m_client->setUsername("formation-sts-sn-supervision");
    m_client->setPassword("ttn-account-v2.5h8uByRog3gRkSIHY9-PUDjeFP0GkvmiFXCKGX4p-
20");

}
```

La méthode `connectToHost()` permet d'établir une connexion au broker du serveur TTN. La méthode `state()` permet de contrôler l'état de cette connexion et le signal `stateChanged()` est envoyé lorsque l'état de la connexion a changé.

- Nous allons donc utiliser le signal `clicked()` du bouton `pbConnexion` (dans le concepteur graphique, clique droit sur le bouton puis `Aller au slot…`) et connecter au slot `on_pbConnexion_clicked()`.
- Dans le constructeur, nous allons connecter le signal `stateChanged()` de l'objet `m_client` à un slot que nous appellerons `stateChange()` dans lequel nous testerons l'état de la connexion et nous afficherons un message en conséquence dans la **barre de status** de l'application.
- Lorsque le status passe à **Connecté**, nous allons aussi souscrire aux messages relatifs aux topic qui contient l'état de la LED et la valeur de la PWM :

`formation-sts-sn-supervision/devices/actionneur-1/up`

```
#define TOPIC_UP "formation-sts-sn-supervision/devices/actionneur-1/up"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    m_client = new QMqttClient(this);
    m_client->setHostname("eu.thethings.network");
    m_client->setPort(1883);
    m_client->setUsername("formation-sts-sn-supervision");
    m_client->setPassword("ttn-account-v2.5h8uByRog3gRkSIHY9-PUDjeFP0GkvmiFXCKGX4p-
20");

    connect(m_client, &QMqttClient::stateChanged, this, &MainWindow::stateChange);
}

void MainWindow::on_pbConnexion_clicked()
{
    if (m_client->state() == QMqttClient::Disconnected) {
        ui->pbConnexion->setText(tr("Deconnexion"));
        m_client->connectToHost();

    } else {
        ui->pbConnexion->setText(tr("Connexion"));
        m_client->disconnectFromHost();
    }
}

void MainWindow::stateChange()
{
    QString message;
    switch (m_client->state())
    {
        case 0 :message = "Déconnecté";break;
        case 1 :message = "En cours de connexion";break;
        case 2 :message = "Connecté";
            QString myTopic1 = TOPIC_UP;
            auto subscription1 = m_client->subscribe(myTopic1);
            if (!subscription1) {
                QMessageBox::critical(this, "Erreur", "Impossible de souscrire au
topic\n"+myTopic1);
                return;
            }
            break;
    }
    ui->statusBar->showMessage(message);
}
```

- Ne pas oublier de déclarer le slot en privé dans le fichier `mainwindow.h` :

```
private slots:
    void on_pbConnexion_clicked();
    void stateChange();
```

En cas de déconnexion

- Ajoutons un slot en réponse au signal `disconnected` envoyé par la classe `QMqttClient` en cas de déconnexion avec le broker :

```
connect(m_client, &QMqttClient::disconnected, this, &MainWindow::brokerDisconnected);
```

- Le solt provoque l'ouverture d'une boite de message :

```
void MainWindow::brokerDisconnected()
{
    QMessageBox::critical(this, "Erreur", "Connexion avec le broker interrompue");
}
```

La réception des messages

La classe `QMqttClient` envoie le signal `messageReceived(message, topic)` lorsqu'un message MQTT arrive du broker.

- Nous allons traiter ce signal dans un slot que nous appellerons `receivedMessage(message, topic)` dans lequel nous lirons la valeur reçue en fonction du nom du topic et nous l'afficherons dans le bon widget.
- De plus, nous afficherons une information furtive dans la barre de status à chaque nouvelle réception d'un message pour s'assurer de la continuité du fonctionnement. C'est important dans le cas de mesures très stables, au moins, on sait qu'on est toujours en vie !

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    m_client = new QMqttClient(this);
    m_client->setHostname("eu.thethings.network");
    m_client->setPort(1883);
    m_client->setUsername("formation-sts-sn-supervision");
    m_client->setPassword("ttn-account-v2.5h8uByRog3gRkSIHY9-PUDjeFP0GkvmiFXCKGX4p-
20");

    connect(m_client, &QMqttClient::disconnected, this,
&MainWindow::brokerDisconnected);
    connect(m_client, &QMqttClient::stateChanged, this, &MainWindow::stateChange);
    connect(m_client, &QMqttClient::messageReceived, this,
&MainWindow::receivedMessage);
}

void MainWindow::receivedMessage(const QByteArray &message, const QMqttTopicName
&topic)
{

    ui->statusBar->showMessage("Receive Data ...",1000);
    QString unit;

    QJsonDocument json = QJsonDocument::fromJson(message);
    QJsonObject obj = json.object();

    QJsonValue payload_fields = obj.value("payload_fields");

    qDebug() << payload_fields;

    int led = payload_fields["led"].toInt();
    int pwm = payload_fields["pwm"].toInt();

    qDebug() << "Led : " << led << " - pwm : " << pwm << "%";

    ui->lbLED->setText(QString::number(led));

    if(pwm>100) pwm = 100;
    ui->progressPWM->setValue(pwm);

}
```

- Ne pas oublier de déclarer le slot privé `receivedMessage()` :

```
private slots:
    void on_pbConnexion_clicked();

    void brokerDisconnected();
    void stateChange();
    void receivedMessage(const QByteArray &message, const QMqttTopicName &topic);
```

Envoi des nouvelles consignes

On envoi les consignes pour contrôler la LED_BUILTIN et la LED PWM en cliquant sur le bouton `pbPublier`. La documentation de l'API MQTT de TTN nous dit :

Figure 29. API MQTT de TTN : messages downlink payload_raw

On remarque que le `payload_raw` qui contient les données à envoyer est encodé en Base64. La métode `toBase64()` de la classe `QByteArray` nous sera très utile.

Le Base64 est un codage de données utilisant 64 caractères (en fait 65 avec le caractère de complément). On forme des groupes de 24 bits successifs de données qui sont codés par une chaîne de 4 caractères. Un caractère représente donc 6 bits. Si on a moins de 24 bits, on ajoute des « = » complémentaires pour former quand même 4 caractères. Chaque groupe de 6 bits est utilisée comme index dans la table de codage.

Binary	ASCII	Binary	ASCII	Binary	ASCII	Binary	ASCII
000000	A	010000	Q	100000	g	110000	w
000001	B	010001	R	100001	h	110001	x
000010	C	010010	S	100010	i	110010	y
000011	D	010011	T	100011	j	110011	z
000100	E	010100	U	100100	k	110100	0
000101	F	010101	V	100101	l	110101	1
000110	G	010110	W	100110	m	110110	2
000111	H	010111	X	100111	n	110111	3
001000	I	011000	Y	101000	o	111000	4
001001	J	011001	Z	101001	p	111001	5
001010	K	011010	a	101010	q	111010	6
001011	L	011011	b	101011	r	111011	7
001100	M	011100	c	101100	s	111100	8
001101	N	011101	d	101101	t	111101	9
001110	O	011110	e	101110	u	111110	+
001111	P	011111	f	101111	v	111111	/

Figure 30. Table de décodage Base64

Merci Thierry !!!

```
void MainWindow::on_pbPublier_clicked()
{
    QByteArray payload;

    QByteArray port = "{\"port\":1,";
    QByteArray payload_raw = "\"payload_raw\":\"";
    QByteArray end = "\"}";
    QByteArray message;

    QString myTopic1 = TOPIC_DOWN;

    int pwm;
    pwm = ui->slidePWM->value();
    pwm = pwm << 1;

    payload[0] =pwm + ui->checkBox->isChecked();

    message.append(port);
    message.append(payload_raw);
    message.append(payload.toBase64());
    message.append(end);
    m_client->publish(myTopic1,message,2);
    qDebug() << message;
    ui->statusBar->showMessage("Send Data ...",1000);
}
```

6.3.6. Test de bout en bout

- Compilez et exécutez votre application.
- Vérifier que vous pouvez bien piloter vos LED. Remarquez que les changements ne sont effectifs qu'après avoir reçu des données de la carte.

Le pilotage n'est pas "instantanné" ! On peut voit dans la console de TTN les données "programmées" (`scheduled`) en attente de pouvoir être expédiées à la carte.

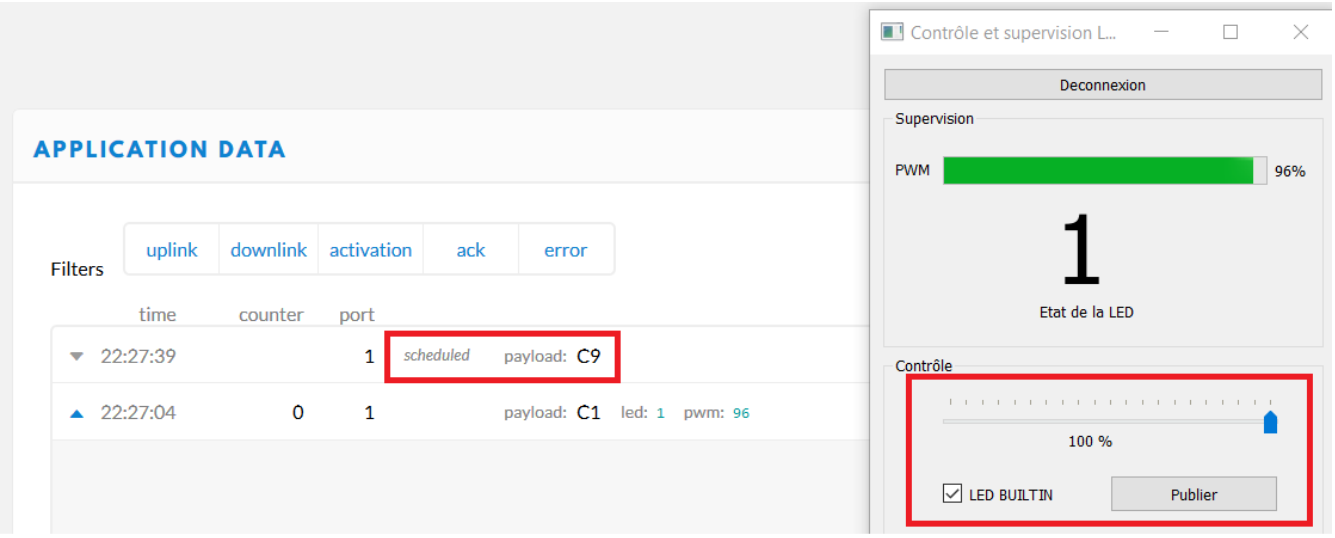


Figure 31. Test de l'application

7. Conclusion

L'utilisation du protocole MQTT depuis Qt n'est pas très compliqué et permet de réaliser des applications de gestion des objets connectés. L'utilisation du format `JSON` est très pratique pour récupérer les données enoyées depuis TTN.

Il est également possible d'utiliser ce format pour envoyer des données de notre application Qt vers TTN. Il suffit d'utiliser `payload_field` plutôt que `payload_raw` :

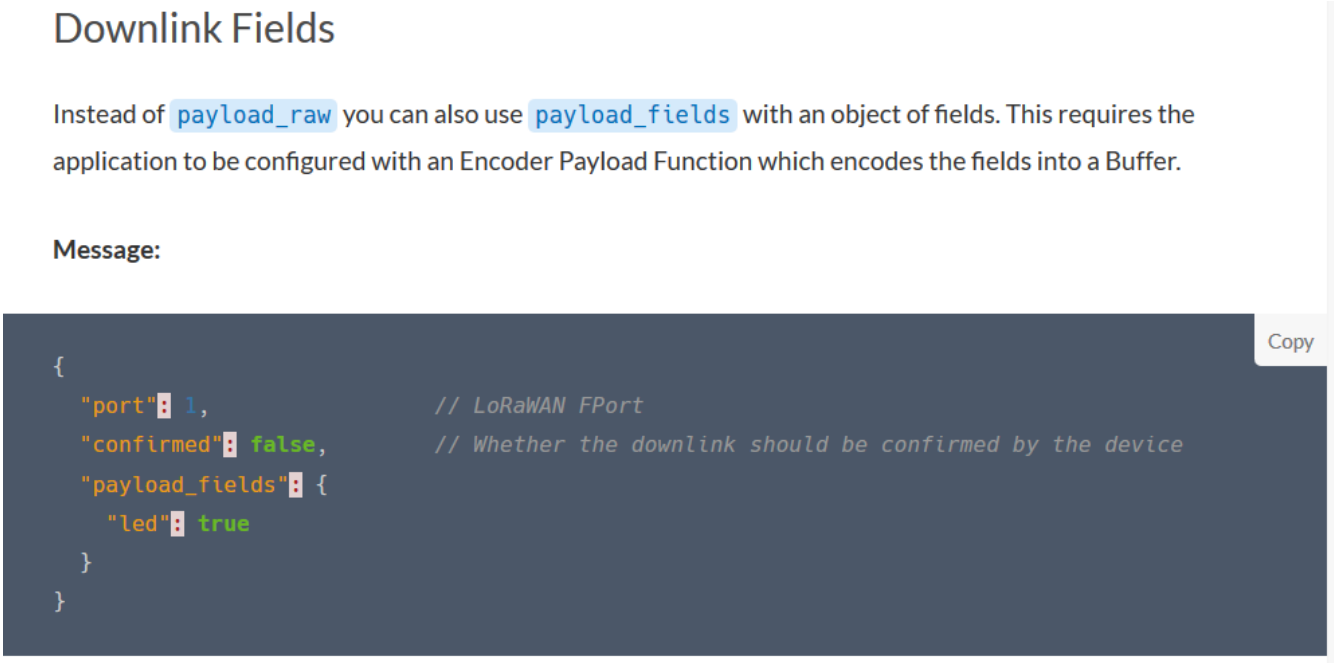


Figure 32. API MQTT de TTN : messages downlink payload_fields

Dans ce cas, il faudra également écrire une fonction d'encodage dans `payload formats` de votre application dans la console de TNN.

```
function Encoder(object, port) {
  // Encode downlink messages sent as
  // object to an array or buffer of bytes.
  var bytes = [];

  if (port === 1) bytes[0] = object.pwm << 1 | object.led;

  return bytes;
}
```