

# JSON API REST MQTT - MQ Telemetry/Transport

Ph. Truillet

Novembre 2023 – v. 2.0



## A. JSON - JavaScript Object Notation

JSON est un format d'échange de données, facile à lire par un humain et interpréter par une machine. Basé sur JavaScript, il est complètement indépendant des langages de programmation mais utilise des conventions qui sont communes à tous les langages de programmation (C, C++, Perl, Python, Java, C#, VB, JavaScript, ...).

Deux structures essentielles sont utilisées par JSON :

- Une collection de clefs/valeurs : **Object**. L'objet commence par un « { » et se termine par « } » et composé d'une liste non ordonnée de paires clefs/valeurs. Une clef est suivie de « : » et les paires clef/valeur sont séparés par « , »
- Une collection ordonnée d'objets : **Array**, liste ordonnée d'objets commençant par « [ » et se terminant par « ] », les objets sont séparés l'un de l'autre par « , »

JSON supporte plusieurs types de données :

- une **Valeur**
  - **Numérique** : nombre entier ou flottant
  - **Chaîne de caractères** : ensemble de caractères Unicode (sauf une double quote " et un antislash \) entouré par des doubles guillemets.
  - **Booléen** : true ou false
  - La valeur **null**
- un **Tableau** : un ensemble ordonné de valeurs entouré par des crochets [ et ]. Chaque valeur est séparée par un caractère virgule. Les types des valeurs d'un tableau peuvent être différents
- un **Objet** : est composé de paires clé/valeur, chacune étant séparée par une virgule, entourées par des accolades { et }.  
Une clé est obligatoirement une chaîne de caractères. Une valeur peut être littérale (chaîne de caractères, un nombre, un booléen, la valeur null), un objet ou un tableau. Une clé est séparée de sa valeur par le caractère « deux points : »

Les valeurs d'un objet ou d'un tableau peuvent être d'un de ces types.

Dans une chaîne de caractères, le caractère d'échappement est le caractère antislash qui permet notamment de représenter dans la chaîne de caractères :

- \" : une double quote
- \\ : un antislash
- \/ : un slash
- \b : un caractère backspace
- \f : un caractère formfeed
- \n : une nouvelle ligne
- \r : un retour chariot
- \t : une tabulation
- \unnnn : le caractère Unicode dont le numéro est nnnn

## Un exemple de JSON

```

{
  "city": {
    "id": 2972315,
    "name": "Toulouse",
    "coord": {
      "lon": 1.4437,
      "lat": 43.6043
    },
    "country": "FR",
    "population": 433055,
    "timezone": 7200
  },
  "cod": "200",
  "message": 0.0492599,
  "cnt": 1,
  "list": [ ... ] // 1 item
}
  
```

Key (clé)

Value (valeur)

### A.1 Exercice JSON et Processing.org

Afin de manipuler des données JSON avec Processing.org, plusieurs opérations sont nécessaires.

1. Charger les données JSON en mémoire  
`JSONObject json = loadJSONObject(chaine_JSON)`

Ici, on charge dans la structure JSON le contenu d'un fichier JSON (souvent récupéré après une requête sur le web)

Il va falloir ensuite accéder à chaque champ en chargeant d'éventuels tableaux ou objets pour y arriver

2. Accéder à un tableau  
 Par exemple, nous pouvons extraire le tableau :  
`JSONArray values = json.getJSONArray("list");`
3. Accéder à un objet  
`JSONObject list = values.getJSONObject(0);`
4. Récupérer une valeur  
`float pressure = list.getFloat("pressure");`

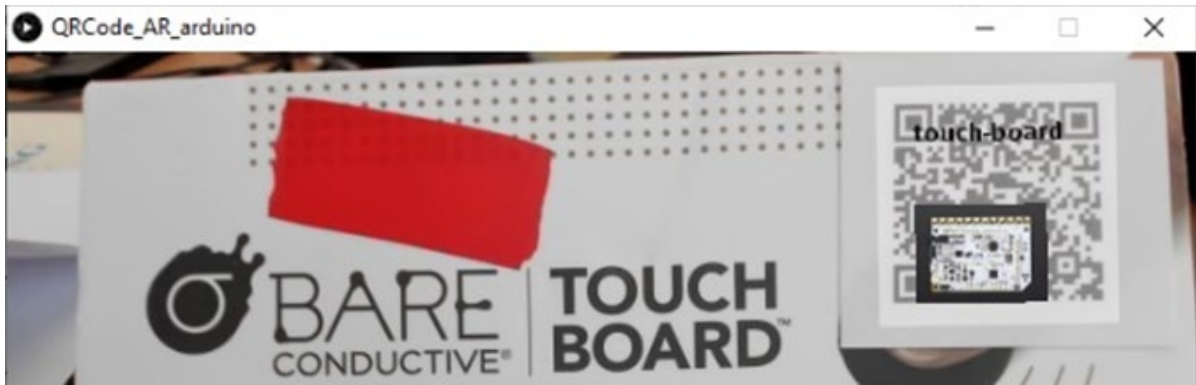
```

    "cnt": 1,
    "list": [
      {
        "dt": 1602154800,
        "sunrise": 1602136805,
        "sunset": 1602177778,
        "temp": { ... }, // 6 items
        "feels_like": { ... }, // 4 items
        "pressure": 1024,
        "humidity": 57,
        "weather": [
          {
            "id": 803,
            "main": "Clouds",
            "description": "broken clouds",
            "icon": "04d"
          }
        ],
        "speed": 1.96,
        "deg": 40,
        "clouds": 58,
        "pop": 0
      }
    ]
  
```

Développer une application Processing.org (cf. Figure 1 plus bas) qui permet de gérer une collection de matériel physiques (exemple : des boîtes de capteurs, arduinos, ...).

Cette application va permettre :

1. de flasher un QRCode codant un contenu JSON. Dans le contenu JSON sera stocké au moins un id, un nom et une url (vers la fiche produit)
2. d'afficher les données décodées sur le QRCode en réalité augmentée. L'url devra enfin être cliquable.



**Figure 1** - Exemple de reconnaissance et d'affichage en réalité augmentée

Vous pouvez partir de l'exemple ci-dessous pour décoder des QR Codes :

<https://github.com/truillet/upssitech/blob/master/SRI/1A/Code/QRCode.zip>

## B. REST – Representational State Transfer

REST (**RE**presentational **S**tate **T**ransfer) n'est pas à proprement parlé un protocole mais un « *style d'architecture* » défendu par Roy Fielding en 2000.

Ce dernier a défini plusieurs contraintes afin d'être conforme à l'architecture REST (« *REST compliant* »)

- Le client (interface utilisateur) et le serveur sont indépendants (stockage, ...)
- Aucune variable de session ou état volatile ne doit être enregistré côté serveur : chaque requête doit être indépendante.
- Le serveur indique au client s'il peut mettre en cache les données qu'il reçoit afin d'éviter les requêtes inutiles et préserver la bande passante.
- Une interface uniforme : chaque ressource est accessible de manière unique.
- Une hiérarchie par couche

A la différence des protocoles RPC (**R**emote **P**rocedure **C**all) et SOAP (**S**imple **O**bject **A**ccess **P**rotocol), **REST** n'impose que peu de contraintes. Les applications respectant cette architecture sont dites **RESTful**.

Les ressources peuvent subir quatre opérations de base : « **CRUD** » (**C**reate, **R**etrieve, **U**ppdate et **D**eleter). REST est souvent utilisé dans un contexte web avec le protocole HTTP en tirant parti du protocole lui-même (mots-clés GET, POST, PUT et DELETE) et de l'utilisation d'URI (Uniform Resource Identifier) comme représentant d'identification des ressources.

L'API peut utiliser n'importe quel moyen de communication pour initier l'interaction entre les applications. Les formats d'échanges entre les clients et le serveur sont la plupart du temps du plaintext, **xml** (**eX**tended **M**arkup **L**anguage) ou **JSON** (**J**avaScript **O**bject **N**otation) définie par la RFC 4627

(<https://tools.ietf.org/html/rfc4627>).

REST a de nombreux avantages comme être **évolutif**, **simple à mettre en œuvre** avec des représentations multiples mais a l'inconvénient de ne garantir qu'une sécurité restreinte par l'emploi des méthodes HTTP.

### B.1. Exercice avec un client http pour consommer les services

Nous utilisons ici le module python requests (`pip install requests`) qui permet d'effectuer des requêtes web à partir d'un fichier python.

Télécharger l'exemple à l'adresse suivante :

<https://github.com/truillet/upssitech/blob/master/SRI/3A/ID/TP/Code/myHttpClient.py>

Décoder les données JSON dans une structure préalablement définie avec un parser JSON (`pip install json`)

## B.2. Lire et traiter du JSON via une API REST

- Créer un compte (*gratuit*) sur *open exchange rates* (<https://openexchangerates.org>) et créer une application (*dans le langage que vous souhaitez*) qui utilise l'API REST proposée pour permettre d'afficher le taux de change entre différentes monnaies (exemple \$US, € et £)
- Avec <https://openweathermap.org>, développer une application qui permet de demander à l'utilisateur **un nom de ville** (dans une interface graphique ou non), faire l'appel nécessaire, récupérer et afficher la météo du jour (icône en png) et la température de la ville concernée.

**Nota :** cet exercice a déjà été proposé dans le cadre de l'initiation à Processing.org en 1<sup>ère</sup> année 😊

## B.3. Produire du contenu JSON avec une API REST

Créer une petite application web « *Annuaire* » qui renvoie une structure JSON contenant les coordonnées complètes de la personne recherchée lorsque l'utilisateur tape une url depuis un navigateur web de type : <http://@ip/searchbyname?name=nom>

**Nota :** vous pouvez par exemple utiliser un framework web en python comme **bottle** (<https://bottlepy.org>) ou **flask** (<https://flask.palletsprojects.com>)

Créer enfin une application cliente (*dans le langage de votre choix*) qui fait les appels nécessaires au serveur et affiche les résultats dans un format « **lisible** ».

## C. MQTT : un protocole pour l'IoT

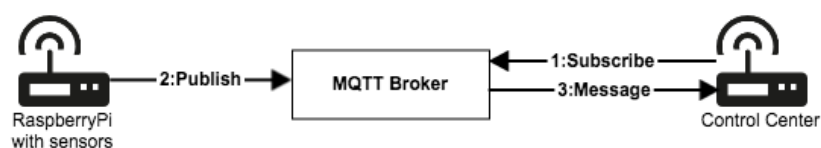
MQTT est un standard ISO (**ISO/IEC PRF 20922 :2016<sup>1</sup>**). Au départ, **TT** (premier nom du protocole) a été développé par Andy Stanford-Clark (IBM) et Arlen Nipper (Eurotech, actuellement Cirrus Link) en 1999 pour le contrôle d'un pipeline dans le désert. L'objectif était d'avoir un protocole de bande passante efficace utilisant peu d'énergie à un moment où les périphériques ne pouvaient être connectés qu'au travers de satellites.

MQTT est un protocole ouvert, simple, léger et facile à mettre en œuvre. Ce protocole est idéal pour répondre aux besoins suivants :

- l'utilisation d'une très faible bande passante,
- l'utilisation sur les réseaux sans fils,
- la consommation en énergie,
- la rapidité avec un temps de réponse supérieur aux autres standards du web actuel,
- la fiabilité,
- et l'usage de faibles ressources processeurs et de mémoire.

Le protocole MQTT utilise une architecture « *publish/subscribe* » en contraste avec le protocole HTTP et son architecture « *request/response* » (cf. Figure 2).

Le point central de la communication est le broker MQTT (qui par défaut utilise le port 1883 pour les connexions non chiffrées et le port 8883 pour celles chiffrées en SSL/TLS), en charge de relayer les messages des émetteurs vers les clients. Chaque client s'abonne via un message vers le broker : le « *topic* » (sorte d'information de routage pour le broker) qui permettra au broker de réémettre les messages reçus des producteurs de données vers les clients. Les clients et les producteurs n'ont ainsi pas à se connaître, ne communiquant qu'au travers des topics. Cette architecture permet des solutions multi-échelles.



**Figure 2 :** Architecture MQTT “Publish/Subscribe”

Chaque client MQTT a une connexion ouverte en permanence avec le broker. Si la connexion s'arrête, le broker bufférise les messages et les émet dès que la reconnexion avec le client est effectuée.

<sup>1</sup> <https://www.iso.org/standard/69466.html>

Un « **topic MQTT** » est une chaîne de caractères qui peut posséder une hiérarchie de niveaux séparés par le caractère « / ». Par exemple, une information de température du salon pourrait être envoyée sur le topic « **maison/salon/temperature** » et la température de la cuisine sur « **maison/cuisine/temperature** ».

**Nota** : Les topics doivent au moins contenir un caractère (ne pas commencer par le caractère « \$ », réservé à un usage interne de MQTT), sont sensibles à la casse et le topic « / » est valide !

### C.1. Wildcards

- Le signe « + » est un caractère « *wildcard* » qui permet des valeurs arbitraires pour un niveau en particulier

**Ex** : « **maison/+/temperature** » permet de s'abonner aux températures de la cuisine et du salon

- le signe « # » pour plus d'un niveau (ce signe doit se trouver à la fin). Les messages envoyés peuvent être de toutes sortes mais ne peuvent excéder une taille de 256 Mo.

**Ex** : si un *publisher* s'abonne au topic « **maison/salon/#** », il recevra toutes les données provenant du salon. De même, s'il s'abonne au topic « **maison/#** », il collectera toutes les données des sondes de la maison.

### C.2. Sécurité

Les données IoT échangées peuvent s'avérer très critiques, c'est pourquoi il est aussi possible de sécuriser les échanges à plusieurs niveaux :

- Transport en SSL/TLS,
- Authentification par certificats SSL/TLS,
- Authentification par login/mot de passe.

### C.3. QoS -Qualité de Service

MQTT intègre en natif la notion de QoS. En effet le *publisher* a la possibilité de définir la qualité de son message. Trois niveaux sont possibles :

- Un message de **QoS niveau 0** « *At most once* » sera délivré tout au plus une fois. Ce qui signifie que le message est envoyé sans garantie de réception, (le broker n'informe pas l'expéditeur qu'il l'a reçu et le message)
- Un message de **QoS niveau 1** « *At least once* » sera livré au moins une fois. Le client transmettra plusieurs fois s'il le faut jusqu'à ce que le Broker lui confirme qu'il a été transmis sur le réseau.
- Un message de **QoS niveau 2** « *exactly once* » sera obligatoirement sauvegardé par l'émetteur et le transmettra toujours tant que le récepteur ne confirme pas son envoi sur le réseau. La principale différence étant que l'émetteur utilise une phase de reconnaissance plus sophistiquée avec le broker pour éviter une duplication des messages (plus lent mais plus sûr).

Par défaut, **MQTT utilise la QoS de niveau 0.**

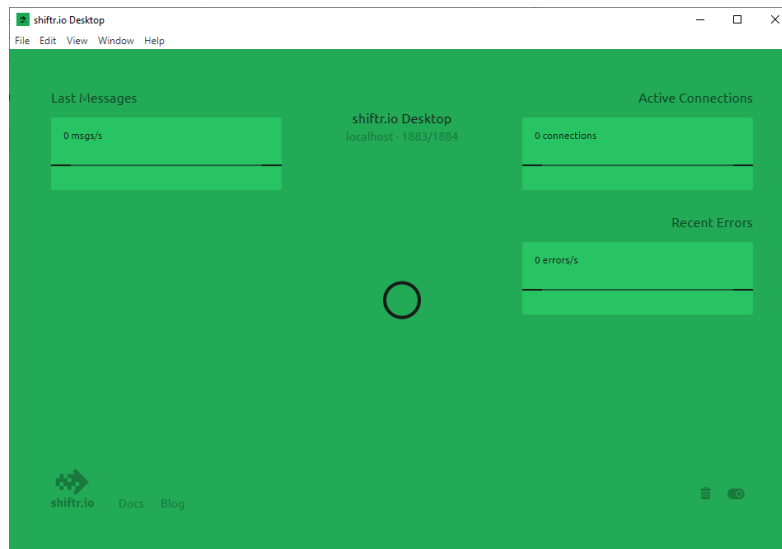
### C.4. Exercices de chauffe

#### C.4.0 Préambule

Pour pouvoir fonctionner, vous allez devoir utiliser un broker MQTT. Vous pouvez facilement en installer un en allant sur le site <https://shiftr.io> et télécharger la **Desktop App** (disponible sous Linux, Windows et MacOS).

**Nota** : si cela ne fonctionne pas, vous pouvez vous rendre sur le site : <https://shiftr.io/try>


Dézippez et lancez le *DesktopApp shiftr.io* : un broker MQTT est lancé sur votre machine, prêt à recevoir vos messages ! Vous devriez voir une fenêtre s'ouvrir (cf. Figure 3)



**Figure 3 : DesktopApp – shiftr.io**

#### C.4.1 Un premier exemple avec Processing.org

Ouvrez Processing, installez la librairie MQTT (Menu « **Outils** » | « **Ajouter un outil** », onglet « **Libraries** » MQTT.

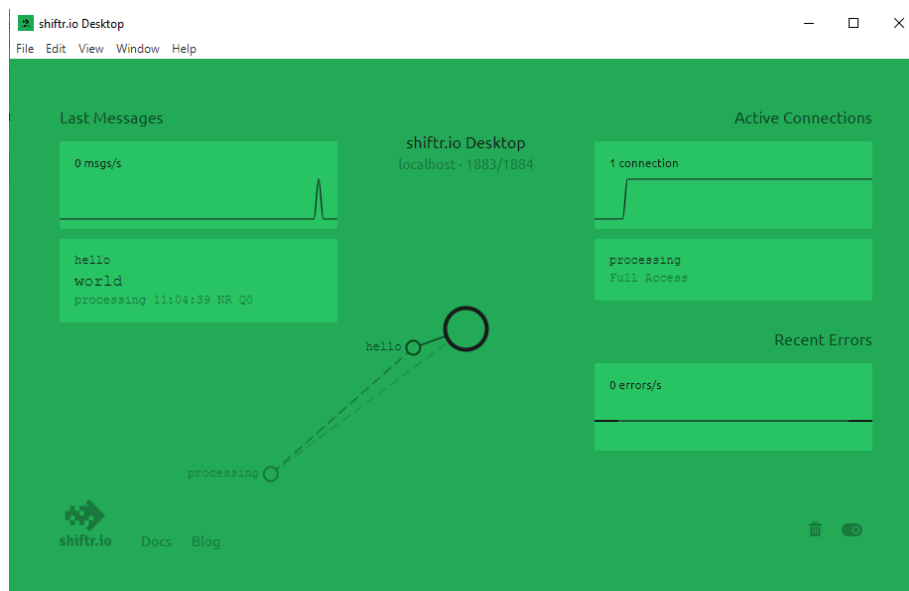
 **MQTT | MQTT library for Processing based on the Eclips...** **Joel Gaehwiler**

Ouvrez l'exemple installé avec la librairie MQTT (Menu « **Fichier** » | « **Exemples...** », | « **Contributed Libraries** » | « **MQTT** », exemple *PublishSubscribe*.

Remplacez l'url utilisée dans l'instruction `client.connect` ligne 20

« `mqtt://try:try@broker.shiftr.io` » par « `mqtt://localhost` »

Lancez le sketch. Appuyez sur la base espace dans le sketch Processing.org et visualisez le résultat sur le DesktopApp (cf. Figure 4)



**Figure 4 : Visualisation des messages MQTT “Publish/Subscribe”**

### Exercice :

- Ecrivez maintenant à partir de cet exemple un programme Processing.org qui génère des valeurs aléatoires de température toutes les secondes et les envoie au broker MQTT (**vous devez déterminer le topic**). Chaque instance lancée sera considérée comme une « pièce » de la maison)
- Ecrivez ensuite un programme qui devra s'abonner à l'ensemble des températures émises par chacune des instances et afficher la moyenne générale des températures de la « maison » (générée donc par chacune des pièces) et les moyennes par « pièce ».

#### C.4.2 Python (Paho)

MQTT est multi-langage et on peut utiliser python comme langage support (intéressant pour un Raspberry Pi par exemple). Pour utiliser MQTT sous Python, allez voir la documentation ici :

<https://pypi.python.org/pypi/paho-mqtt>

- Ecrivez le même programme émetteur de données en Python.

#### C.4.3 Arduino

De la même manière, il est possible de générer des messages MQTT depuis un périphérique compatible arduino connecté à TCP/IP (via un réseau filaire ou en WiFi)

Il faut dans un premier temps récupérer la librairie MQTT en faisant « **Croquis | Importer une bibliothèque | Gérer les bibliothèques** » et chercher la bibliothèque **MQTT** (Arduino 1.8.7 et ultérieur)



Vous trouverez dans la section « *Exemples* » du code permettant d'émettre des données depuis un module arduino directement sur MQTT.

#### C.4.4 Un broker MQTT

Vous avez accès au broker mosquitto MQTT en ligne pour vos tests à l'adresse IP **mqtt.upssitech.fr** sur le port 1883 (pas de login/password).

Le broker MQTT le plus connu et le plus utilisé reste **Mosquitto** (géré par la fondation **eclipse**)

Vous pouvez télécharger **Mosquitto** ici : <https://mosquitto.org>



## D. Un Superviseur de capteurs – Tout lier pour tout contrôler !

Vous allez maintenant devoir créer, de gérer et superviser un réseau de capteurs à partir de données « **réelles locales** » et à distance (en utilisant par exemple des données récupérées via une **API REST**). Pour ce faire :

1. créez un client MQTT qui récupère les données de la météo locale sur **OpenWeatherMap** et les envoie sur le broker.
2. créez des clients MQTT à partir de micro-contrôleurs compatibles **arduino** ou **Raspberry Pi** et de capteurs de luminosité, température (ou autre) des sondes ... et générez des valeurs vers le broker MQTT (vous pourrez le simuler si vous n'avez pas le matériel à disposition).
3. Ecrivez un programme tiers capable de lire des données du broker, agréger les valeurs des différents capteurs positionnés dans l'environnement et renvoyez périodiquement ces valeurs sur le broker (déterminez les topics à utiliser).
4. proposez enfin une visualisation (type évolution des valeurs dans le temps) de vos données

## Liens

- Introducing JSON, <https://www.json.org/json-en.html>
- Bonnes pratiques pour développer des APIs REST, <https://www.gekko.fr/les-bonnes-pratiques-a-suivre-pour-developper-des-apis-rest>