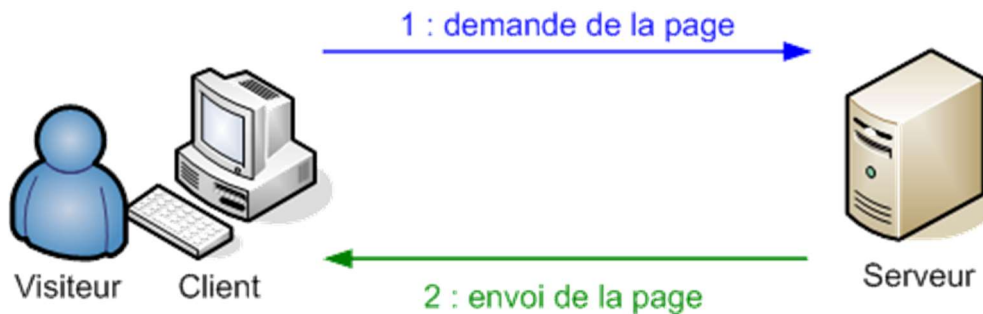


1 – Présentation du protocole HTTP

Le Hypertext Transfer Protocol, plus connu sous l'abréviation HTTP, littéralement le « protocole de transfert hypertexte », est un protocole de communication client-serveur développé pour le World Wide Web. Il est utilisé pour échanger toute sorte de données entre client HTTP et serveur HTTP.



Les principales méthodes de requête client sont les méthodes GET et POST.

La méthode GET transmet les données dans l'entête et la méthode POST dans le corps.

Les capteurs IO LINK de IFM acceptent à priori les 2.

Contexte du TP : Un maître IO LINK AL1320 est connecté au réseau par le port IoT Port. Un capteur de vibration IO LINK VBB001 (version B) est connecté sur le port 1 (X01) du maître.

2 – Premiers essais de requêtes http

Requête GET avec un explorateur internet

Remarque : Les requêtes http utilisent le **port 80** par défaut.

⇒ Faire une recherche de l'adresse IP du Maître IO LINK AL1320 ⇒ adr_IP

⇒ Sous l'explorateur internet (Firefox par exemple) taper la commande suivante :

`http://adr_IP:80/deviceinfo/vendor/getdata` ou `http://adr_IP/deviceinfo/vendor/getdata`

⇒ Observer le fichier JSON renvoyé par le serveur, ici le AL1320.

```
JSON  Données brutes  En-têtes
Enregistrer Copier Formater et indenter

{
  "cid": -1,
  "data": {
    "value": "ifm electronic gmbh"
  },
  "code": 200
}
```

Requête GET avec NODE RED

⇒ Réaliser le FLOW ci-dessous et tester.

The screenshot shows a Node-RED flow with three nodes: 'timestamp', 'http request', and 'debug 1'. The 'http request' node is configured with the following settings:

- Method: GET
- URL: 172.16.6.156/deviceinfo/vendor/getdata
- Payload: Ignore
- Enable secure (SSL/TLS) connection:
- Use authentication:
- Enable connection keep-alive:
- Use proxy:
- Only send non-2xx responses to Catch node:
- Disable strict HTTP parsing:
- Return: a parsed JSON object (circled in red)

The 'debug 1' node shows the following output:

```
22/03/2023 18:38:34 node: debug 1
msg.payload : Object
  object
    cid: -1
    data: object
      value: "ifm electronic gmbh"
    code: 200
```

URL : 172.16.6.156/deviceinfo/vendor/getdata
Modifier l'adresse IP du maître IOLINK si différente

Requête POST avec NODE RED

⇒ Réaliser le FLOW ci-dessous et tester.

The screenshot shows a Node-RED flow with four nodes: 'timestamp', 'Vendor_getdata', 'http request', and 'debug 1'. The 'Vendor_getdata' node is configured with the following settings:

- Name: Vendor_getdata
- Setup:
- On Start:
- On Message:

```
1 msg.payload={
2   "code": "request",
3   "cid": 1,
4   "adr": "/deviceinfo/vendor/getdata"
5 }
6 return msg;
```

The 'http request' node is configured with the following settings:

- Method: POST
- URL: 172.16.6.156
- Enable secure (SSL/TLS) connection:
- Use authentication:
- Enable connection keep-alive:
- Use proxy:
- Only send non-2xx responses to Catch node:
- Disable strict HTTP parsing:
- Return: a parsed JSON object

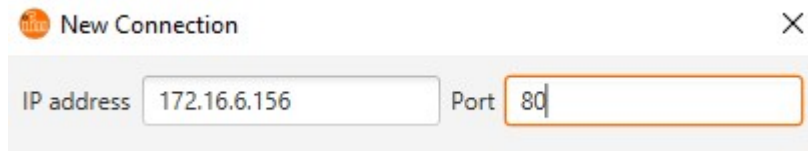
Requête à copier :

```
{
  "code": "request",
  "cid": 1,
  "adr": "/deviceinfo/vendor/getdata"
}
```

3 – Utilisation du logiciel « IoT-Core Assistant_V1.2 » pour interroger le maître IOLINK et le capteur IO LINK

⇒ Lancer le logiciel

⇒ Connecter le logiciel au maître IOLINK : Connection / New Connection – Entrer l'adresse IP et le port 80



New Connection

IP address 172.16.6.156 Port 80

⇒ Retrouver la requête (POST) pour le nom du vendeur et le fichier JSON envoyé au serveur (Master IOLINK).



deviceinfo (structure)

- productcode (data)
- serialnumber (data)
- swrevision (data)
- bootloaderrevision (data)
- fieldbustype (data)
- hwrevision (data)
- vendor (data)
- getdata (service)**
- devicefamily (data)

getdata (service) /deviceinfo/vendor/getdata

Data Input

Method: POST GET

CID: 1

Format: namespace: json, type: string, encoding: UTF-8


Execute Open Response

Live Preview

URL: http://172.16.6.156:80

JSON data: { "code": "request", "cid": 1, "adr": "/deviceinfo/vendor/getdata" }

⇒ Retrouver la requête (GET) pour le nom du vendeur. La requête est ici intégrée à l'adresse.



deviceinfo (structure)

- productcode (data)
- serialnumber (data)
- swrevision (data)
- bootloaderrevision (data)
- fieldbustype (data)
- hwrevision (data)
- vendor (data)
- getdata (service)**
- devicefamily (data)

getdata (service) /deviceinfo/vendor/getdata

Data Input

Method: POST GET

CID: 1

Format: namespace: json, type: string, encoding: UTF-8

Execute Open Response

URL: http://172.16.6.156:80/deviceinfo/vendor/getdata

⇒ Exécuter la requête et retrouver la réponse du serveur (ici en POST).



Time: 22.03.2023 - 19:20:28.941 Method: POST

HTTP status code: 200 - OK IoT-Core code: 200 - OK

URL: http://172.16.6.156:80

JSON Request: { "code": "request", "cid": 1, "adr": "/deviceinfo/vendor/getdata" }

JSON Response: { "cid": 1, "data": { "value": "ifm electronic gmbh" }, "code": 200 }

De la même manière :

⇒ Retrouver les données « process entrée/sortie » (identique à celles qu'on récupère en MQTT), par la méthode POST.

The screenshot shows a REST client interface. On the left, a tree view displays the API structure:

- iolinkmaster (structure)
 - port[1] (structure)
 - senddatatosmob (data)
 - mastercycletime_actual (data)
 - mastercycletime_preset (data)
 - portevent (data)
 - mode (data)
 - comspeed (data)
 - validation_datastorage_mode (data)
 - validation_vendorid (data)
 - validation_deviceid (data)
 - datastorage (data)
 - iolinkdevice (structure)
 - vendorid (data)
 - deviceid (data)
 - applicationspecificitag (data)
 - pdin (data)
 - getdata (service)** (circled in red)
 - iolreadacyclic (service)
 - iolwriteacyclic (service)

On the right, the JSON Request and Response are displayed:

JSON Request

```
{
  "code": "request",
  "cid": 1,
  "adr": "/iolinkmaster/port[1]/iolinkdevice/pdin/getdata"
}
```

JSON Response

```
{
  "cid": 1,
  "data": {
    "value": "0000FC000002FF000000FF000129FF000028FF01"
  },
  "code": 200
}
```

⇒ Retrouver le nom et la version du capteur VBB001 (ici version B)

The screenshot shows a REST client interface. On the left, a tree view displays the API structure:

- iolinkmaster (structure)
 - port[1] (structure)
 - senddatatosmob (data)
 - mastercycletime_actual (data)
 - mastercycletime_preset (data)
 - portevent (data)
 - mode (data)
 - comspeed (data)
 - validation_datastorage_mode (data)
 - validation_vendorid (data)
 - validation_deviceid (data)
 - datastorage (data)
 - iolinkdevice (structure)
 - vendorid (data)
 - deviceid (data)
 - applicationspecificitag (data)
 - pdin (data)
 - iolreadacyclic (service)
 - iolwriteacyclic (service)
 - productname (data)
 - getdata (service)** (circled in red)

On the right, the JSON Request and Response are displayed:

JSON Request

```
{
  "code": "request",
  "cid": 1,
  "adr": "/iolinkmaster/port[1]/iolinkdevice/productname/getdata"
}
```

JSON Response

```
{
  "cid": 1,
  "data": {
    "value": "VVB001 Status B"
  },
  "code": 200
}
```

Lecture et écriture de donnée spécifiée dans la documentation du capteur IO LINK

⇒ Retrouver dans la documentation du capteur VBB001 (B) l'extrait ci-dessous :

7.4 Mémoire

7.4.1 Température

Lo.T	Indice 567	Subindex 0	IntegerT (16 Bit)	ReadOnly
Mémoire valeur minimum pour la température				
Plage de valeurs [°C]	(-300 to 800) * 0.1			
	-32760	(UL)		
	32760	(OL)		
	-32762	(cr.UL)		
	32762	(cr.OL)		
	32764	(NoData)		

Hi.T	Indice 566	Subindex 0	IntegerT (16 Bit)	ReadOnly
Mémoire valeur maximum pour la température				
Plage de valeurs [°C]	(-300 to 800) * 0.1			
	-32760	(UL)		
	32760	(OL)		
	-32762	(cr.UL)		
	32762	(cr.OL)		
	32764	(NoData)		

Dans la mémoire du capteur est mémorisé, dans cet exemple, la valeur min. et la valeur max. atteint par la température.

Avec IoT Core Assistant , on utilise `iolreadacyclic` et `iolwriteacyclic` pour lire et écrire des données spécifiques

- iolinkmaster (structure)
 - port[1] (structure)
 - senddatatosmob (data)
 - mastercycletime_actual (data)
 - mastercycletime_preset (data)
 - portevent (data)
 - mode (data)
 - comspeed (data)
 - validation_datastorage_mode (data)
 - validation_vendorid (data)
 - validation_deviceid (data)
 - datastorage (data)
 - iolinkdevice (structure)
 - vendorid (data)
 - deviceid (data)
 - applicationspecifictag (data)
 - pdin (data)
 - iolreadacyclic (service)**
 - iolwriteacyclic (service)**

Pour la lecture

iolreadacyclic (service) /iolinkmaster/port[1]/iolinkdevice/iolreadacyclic

Data Input

Method: POST GET

CID:

Index:

Subindex:

Open Response

JSON Request

```
{
  "code": "request",
  "cid": 1,
  "adr": "/iolinkmaster/port[1]/iolinkdevice/iolreadacyclic",
  "data": {
    "index": 566,
    "subindex": 0
  }
}
```

JSON Response

```
{
  "cid": 1,
  "data": {
    "value": "012E"
  },
  "code": 200
}
```

0x012E=302 soit 30,2°C (coef de 0,1)

Pour l'écriture :

4 System Commands



System Command information
- Address: Index 2, Subindex 0
- Datatype: UInteger (8 Bit)
- AccessRight: Write Only

System Commands	Text	Description
1	Upload Start	Start block parameter upload
2	Upload End	End block parameter upload
3	Download Start	Start block parameter download
4	Download End	Stop block parameter download
5	Store	Finalize block parameterization and start Data Storage
6	Break	Cancel block parameterization
130	Sélectionner le réglage usine	
165	Remise à zéro mémoires [Hi.T] et [Lo.T]	
166	Remise à zéro mémoire [Lo.T]	
167	Remise à zéro mémoire [Hi.T]	

Avec IoT Core Asistant : **0xA7 = 167**

iolwriteacyclic (service) /iolinkmaster/port[1]/iolinkdevice/iolwriteacyclic

Data Input

Method: POST GET

CID:

Index:

Subindex:

New Value as

HEX-String:

Open Response

JSON Request

```
{
  "code": "request",
  "cid": 1,
  "adr": "/iolinkmaster/port[1]/iolinkdevice/iolwriteacyclic",
  "data": {
    "index": 2,
    "subindex": 0,
    "value": "A7"
  }
}
```

JSON Response

```
{
  "cid": 1,
  "code": 200
}
```

4 – Exploitation avec Node Red

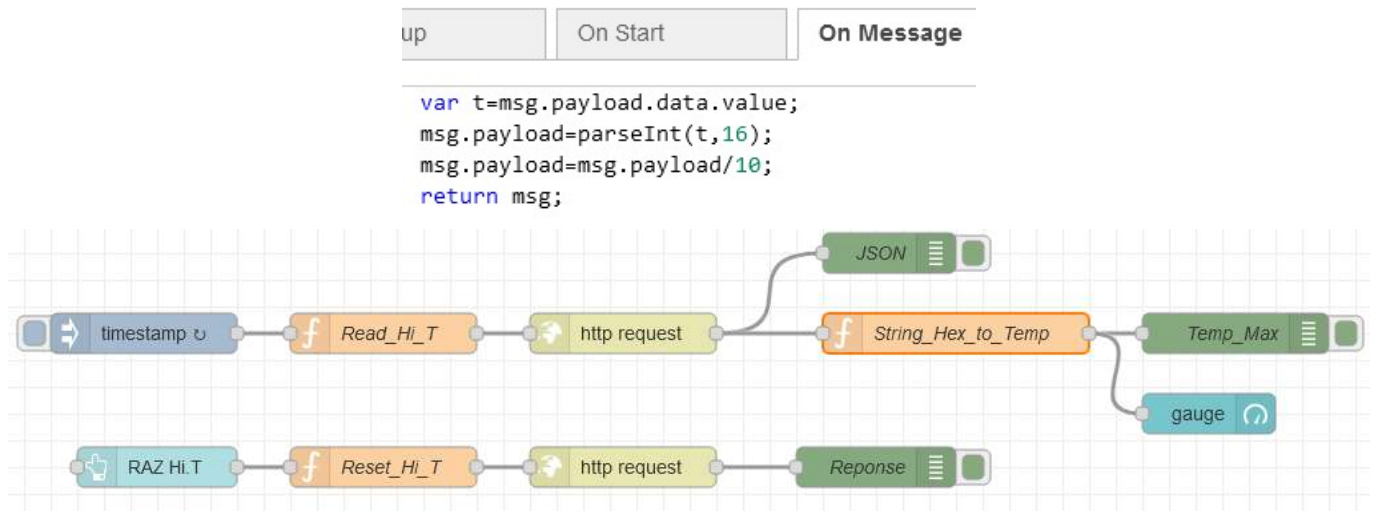
On souhaite réaliser l'interface suivant :



Afficher une jauge affichant la température maximale du capteur enregistrée et avoir un bouton « RAZ Hi.T » pour remettre à zéro cette valeur mémorisée.

⇒ Réaliser le FLOW suivant :

- timestamp sera configuré pour faire une requête toutes les 10 s.
- la fonction String_Hex_to_Temp convertit la chaîne de caractère représentant un nombre HEX en Température.



⇒ De la même manière, réaliser un flow pour lire la valeur maximale de a-RMS (Hi – a-RMS) dans la mémoire du capteur et la remettre à zéro.