Sensor

Plan

- Modèle UDM
  - Présentation du modèle

- Du module à Sensor
  - Exemple d'utilisation d'un module TD12XX : monitoring de batterie
  - Transformation & relais de l'information du module jusqu'à Sensor

- De Sensor vers l'infini et au-delà !
  - Relais par accès API / Relais par callback
  - Modèle IOT
  - Developper API / Device API / Application API

- Sensor
  - Architecture fonctionnelle
  - Architecture technique

- TP

Modèle UDM

- Modèle évolutif permettant de gérer tout type de *device* de manière optimisée

  • 12 événements sont actuellement identifiés, possibilité d'étendre jusqu'à 255

  ✓ *EVENT_BATTERY_LOW* & *EVENT_BATTERY_OK*
  ✓ *EVENT_CONNECTION_LOST* & *EVENT_CONNECTION_OK*
  ✓ *EVENT_RSSI_LOW* & *EVENT_RSSI_OK*
  ✓ *EVENT_SWITCH_ON* & *EVENT_SWITCH_OFF*
  ✓*EVENT_TEMP_LOW* & *EVENT_TEMP_HIGH* & *EVENT_TEMP_OK*
  ✓*EVENT_BOOT*

  • 7 types de messages sont gérés, pouvant aller jusqu'à 15

  ✓ *SRV_FRM_EVENT*
  ✓*SRV_FRM_DATA*
  ✓*SRV_FRM_REGISTER*
  ✓*SRV_FRM_KEEPALIVE*
  ✓*SRV_FRM_RAW*
  ✓*SRV_FRM_SERVICE*
  ✓*SRV_FRM_GEOLOC*

  • permet de constituer des regroupements de 16 *devices*

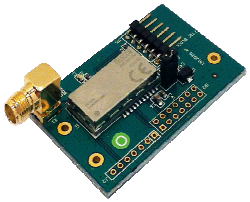
connectingdevices TELECOMDESIGN

Modèle UDM

- Permet de bénéficier de toute la puissance de Sensor avec la structuration du message.
  - Une trame UDM est constituée d'un *header* (2 octets) et d'une *payload* (au plus 10 octets).
  - Le *header* véhicule :
    - ✓ un indicateur *retry* : indique une réémission de trame à l'initiative du module
    - ✓ un *stamp* : compteur de type de trame
    - ✓ un compteur *cpt* : à chaque émission d'une trame par le module, *cpt* est incrémenté
    - ✓ un *type* : identifie la nature de la trame (ex : *SRV_FRM_EVENT* ou *SRV_FRM_KEEPALIVE* …)
    - ✓ un *entry_id :* indique l'emplacement dans la table d'appairage de la passerelle du module communicant (l'*entry_id* de la passerelle vaut toujours 0)
  - La *payload* quant à elle permet le transport de données propres à chaque type de trame.

- Permet d'avoir un langage commun entre les couches basses et hautes pour véhiculer une information.

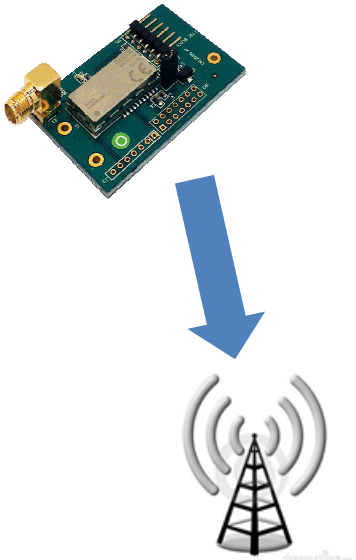
Du module à Sensor



Exemple d'utilisation : monitoring de la batterie
- Activation du monitoring de la batterie
- Lorsqu'une tension faible est détectée par le module TD12XX, un événement de type BATTERY_LOW est encodé selon le modèle UDM. Une fois encodé, ce message ressemble à « 6c000000 ».

connectingdevices TELECOMDESIGN

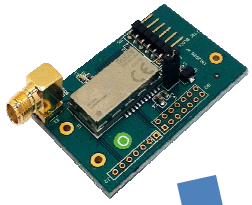
Du module à Sensor



Exemple d'utilisation : monitoring de la batterie
- Activation du monitoring de la batterie
- Lorsqu'une tension faible est détectée par le module TD12XX, un événement de type BATTERY_LOW est encodé selon le modèle UDM. Une fois encodé, ce message ressemble à « 6c000000 ».
- Une trame « 6c000000 » est alors émise sur le réseau Sigfox par le module TD12XX.

connectingdevices TELECOMDESIGN

Du module à Sensor



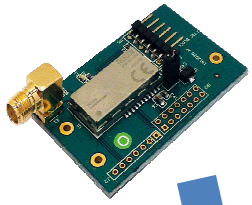
SENSOR

Exemple d'utilisation : monitoring de la batterie

- Activation du monitoring de la batterie
- Lorsqu'une tension faible est détectée par le module TD12XX, un événement de type BATTERY_LOW est encodé selon le modèle UDM. Une fois encodé, ce message ressemble à « 6c000000 ».
- Une trame « 6c000000 » est alors émise sur le réseau Sigfox par le module TD12XX.
- Une fois cette trame reçue par Sigfox, elle est alors relayée sur Sensor avec d'autres informations :
    - l'identifiant du module Sigfox à l'origine de l'émission de la trame
    - l'heure précise au moment de relais vers Sensor (format *timestamp*)
    - l'identifiant de la station relayant la trame
    - l'intensité du signal de couverture Sigfox du module
- Ce relais est sécurisé par un *login / password* + utilisation du protocole HTTPS.

connectingdevices TELECOMDESIGN

Du module à Sensor



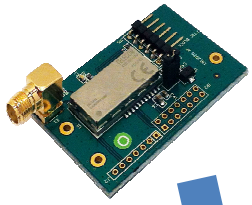
SENSOR

Exemple d'utilisation : monitoring de la batterie

- Activation du monitoring de la batterie
- Lorsqu'une tension faible est détectée par le module TD12XX, un événement de type BATTERY_LOW est encodé selon le modèle UDM. Une fois encodé, ce message ressemble à « 6c000000 ».
- Une trame « 6c000000 » est alors émise sur le réseau Sigfox par le module TD12XX.
- Une fois cette trame reçue par Sigfox, elle est alors relayée sur Sensor avec d'autres informations :
    - l'identifiant du module Sigfox à l'origine de l'émission de la trame
    - l'heure précise au moment de relais vers Sensor (format *timestamp*)
    - l'identifiant de la station relayant la trame
    - l'intensité du signal de couverture Sigfox du module
- Ce relais est sécurisé par un *login / password* + utilisation du protocole HTTPS.
- Sensor effectue alors plusieurs contrôles sur ce *callback* :
    - contrôle de non nullité de la trame
    - contrôle d'existence du module Sigfox dans la base de donnée Sensor (sinon création)
    - contrôle que Sigfox ne nous a pas déjà relayé cette paire (timestamp ; data) (sinon erreur 400)
    - contrôle que « ce message » (non pas « 6c000000 » mais bien l'information de batterie faible n°X par ce module là) n'ait pas déjà été traitée par Sensor => ce contrôle nécessite une phase de décodage de la trame « 6c000000 » selon le modèle UDM. Ainsi, on décode toutes les variables contenues dans la trame (y compris le compteur d'événements X).
- A l'issue de tous ces contrôles, Sensor sauvegarde en base de données une publication contenant :
    - le *timestamp* Sigfox + la trame « 6c000000 » + l'intensité du signal + la station
- Sensor sauvegarde également d'autres informations permettant de :
    - tracer que des trames ont été perdues (utilisation de compteurs véhiculés dans la trame)
    - tracer que des trames dupliquées ont été détectées (variable *hitcount*)
- Si le message véhiculé par la trame nécessite une action de la part de Sensor (ex : envoie de SMS), l'action est alors déclenchée.

connectingdevices TELECOMDESIGN

Du module à Sensor



SENSOR

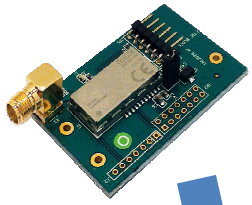
Suivant le type de message un profil de répétition est utilisé par le module TD12XX pour réémettre la même information au bout d'un laps de temps donné.

Dans notre exemple de monitoring de la batterie, le message BATTERY_LOW a été converti en « 6c000000 ».
Un décodage UDM de cette trame donne les variables suivantes :

- retry=**false**
- stamp=6
- cpt=**12**
- type=SRV_FRM_EVENT
- entry_id=0

connectingdevices TELECOMDESIGN

Du module à Sensor



SENSOR

Suivant le type de message un profil de répétition est utilisé par le module TD12XX pour réémettre la même information au bout d'un laps de temps donné.

Dans notre exemple de monitoring de la batterie, le message BATTERY_LOW a été converti en « 6c000000 ».
Un décodage UDM de cette trame donne les variables suivantes :

- retry=**false**
- stamp=6
- cpt=**12**
- type=SRV_FRM_EVENT
- entry_id=0

Deux minutes plus tard, le module réémettra le même message mais avec l'indicateur de *retry* de positionné ainsi qu'une valeur de cpt incrémentée (ie retry=**true** et cpt=**13**).
Cela donne lieu à la trame : « ed000000 ».

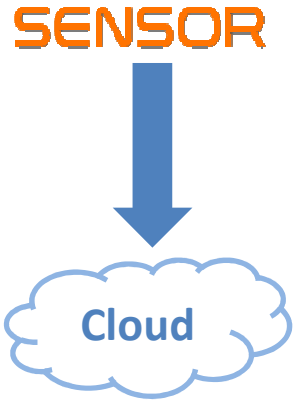
Ce type de trame est alors répété au total 3 fois, espacées chacune de 2 minutes.

La trame « 6c000000 » a été répétée en « ed000000 », elle-même répétée en « ee000000 » et ensuite en « ef000000 ».

Sensor a alors toute la logique nécessaire à la détection de doublons et ne traitera alors que la première trame « 6c000000 » (ie stockage en base, action éventuelle, relais éventuel). Les autres trames seront considérées comme des *duplicate* et ne seront pas sauvegardées en base. Néanmoins, on tracera leur détection.

connectingdevices TELECOMDESIGN

De Sensor vers l'infini et au-delà !



Une fois la publication (ie *timestamp* Sigfox + la trame « 6c000000 » + l'intensité du signal + la station) stockée dans les bases Sensor, une opération d'analyse est armée.

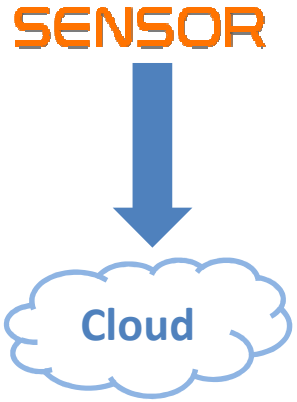
L'opération d'analyse consiste à décoder toutes les paires (variable ; valeur) contenues dans une publication.
L'analyse de la publication ci-dessus donne :

Données de publication

Matériel	Module	Variable	Type	Date	Valeur
1FA0	SFX Module	Station	data	2013-04-09 09:25:01	003B
1FA0	SFX Module	Signal	data	2013-04-09 09:25:01	32.4
1FA0/0	MCS Module	Type	data	2013-04-09 09:25:01	SRV_FRM_EVENT
1FA0/0	MCS Module	Stamp	data	2013-04-09 09:25:01	6
1FA0/0	MCS Module	Retry	data	2013-04-09 09:25:01	false
1FA0/0	MCS Module	EntryId	data	2013-04-09 09:25:01	0
1FA0/0	MCS Module	Cpt	data	2013-04-09 09:25:01	12
1FA0/0	MCS Module	BatteryLow	evt	2013-04-09 09:25:01	EVENT_BATTERY_LOW

connectingdevices TELECOMDESIGN

De Sensor vers l'infini et au-delà !



Une fois l'analyse de la publication effectuée, une opération de relais est alors armée.

L'opération de relais consiste en :
- Rechercher si le module à l'origine de l'émission du message fait partie d'un regroupement (on parle d'une *IOT Application*).
- On itère sur chacune des *IOT Application* ainsi trouvée, et l'on regarde si une adresse de *callback* est mentionnée. Si oui, on relais un objet *IOT Callback* sur cette adresse.

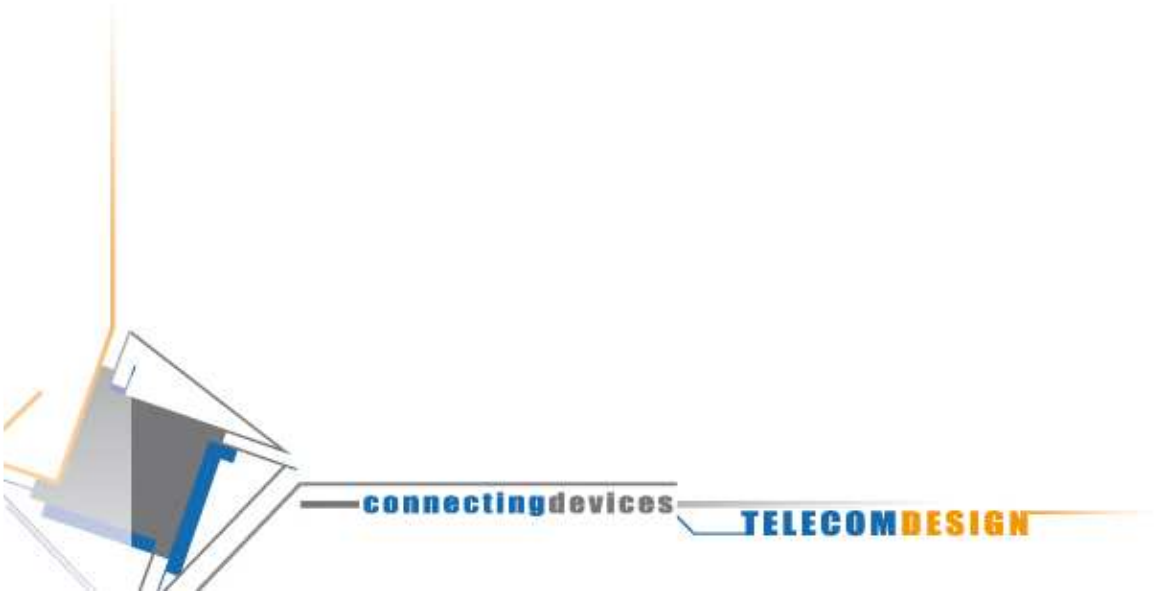
Les protocoles autorisés par Sensor pour faire du relais sont :
- FTP / FTPS
- HTTP / HTTPS
- SMS / EMAIL

De Sensor vers l'infini et au-delà !

Le modèle IOT est utilisé pour relayer un message de Sensor vers un autre SI.

Relayer les trames UDM sérialisées n'a aucun sens, mieux vaut utiliser un modèle objet plus parlant contenant l'interprétation des trames UDM.

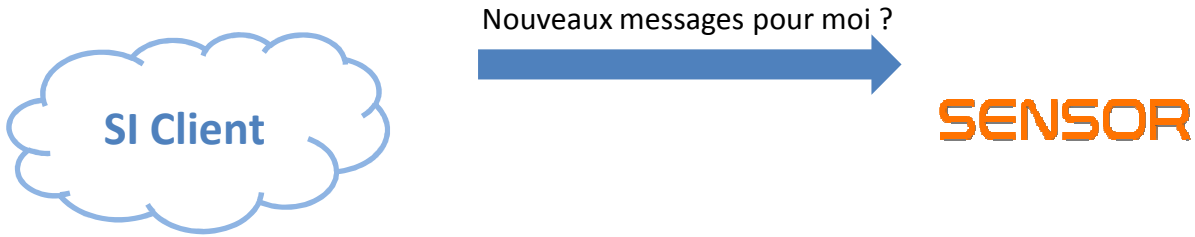
Sensor autorise 2 modes d'accès aux données pour un client :
- Sensor transmet directement le message au client dès qu'il apparait (par callback)
- ou alors, c'est à l'initiative de client de demander à Sensor si des nouveaux messages sont disponibles (par API).



De Sensor vers l'infini et au-delà !

Relais par accès API :

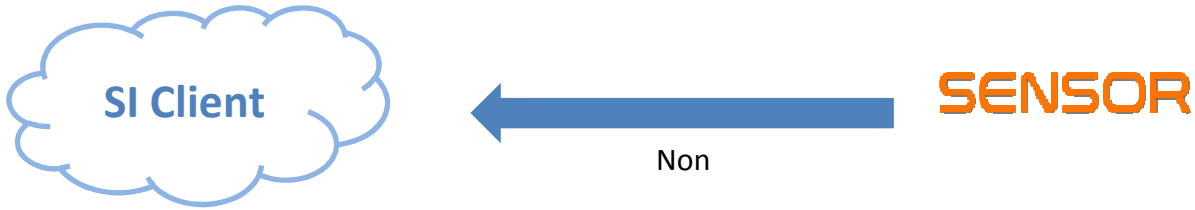
Création côté client d'un mécanisme itératif de consultation des messages disponibles sur Sensor.
S'il n'y a pas de nouveaux messages, on se rendort pour une durée définie par le client (ex : toutes les minutes).
S'il y a un nouveau message, on le prend et on le digère (stockage côté client, analyse, représentation graphique, relais, …).



De Sensor vers l'infini et au-delà !

Relais par accès API :

Création côté client d'un mécanisme itératif de consultation des messages disponibles sur Sensor.
S'il n'y a pas de nouveaux messages, on se rendort pour une durée définie par le client (ex : toutes les minutes).
S'il y a un nouveau message, on le prend et on le digère (stockage côté client, analyse, représentation graphique, relais, …).



De Sensor vers l'infini et au-delà !

Relais par accès API :

Création côté client d'un mécanisme itératif de consultation des messages disponibles sur Sensor.
S'il n'y a pas de nouveaux messages, on se rendort pour une durée définie par le client (ex : toutes les minutes).
S'il y a un nouveau message, on le prend et on le digère (stockage côté client, analyse, représentation graphique, relais, …).

Attente

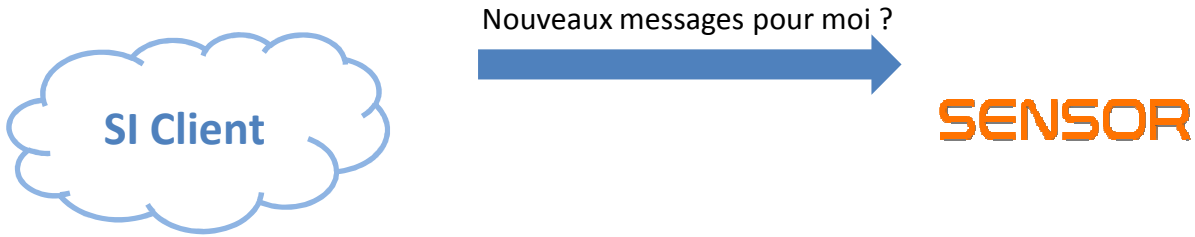
SENSOR

connectingdevices TELECOMDESIGN

De Sensor vers l'infini et au-delà !

Relais par accès API :

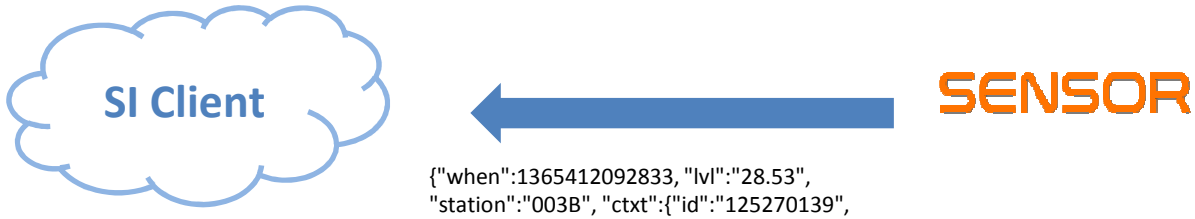
Création côté client d'un mécanisme itératif de consultation des messages disponibles sur Sensor.
S'il n'y a pas de nouveaux messages, on se rendort pour une durée définie par le client (ex : toutes les minutes).
S'il y a un nouveau message, on le prend et on le digère (stockage côté client, analyse, représentation graphique, relais, …).



De Sensor vers l'infini et au-delà !

Relais par accès API :

Création côté client d'un mécanisme itératif de consultation des messages disponibles sur Sensor.
S'il n'y a pas de nouveaux messages, on se rendort pour une durée définie par le client (ex : toutes les minutes).
S'il y a un nouveau message, on le prend et on le digère (stockage côté client, analyse, représentation graphique, relais, …).



{"when":1365412092833, "lvl":"28.53",
"station":"003B", "ctxt":{"id":"125270139",
"uid":"00001234", "serial":"1FA0/1", "lastseen":"Apr 8,
2013 9:08:12 AM", "firstseen":"Apr 8, 2013 9:03:22
AM", "active":true, "watched":false, "type":"OTHER",
"category":"0000", "index":1, "battery":"ON",
"network":"UNKNOWN", "tamper":"UNKNOWN",
"level":"UNKNOWN", "temp":"UNKNOWN",
"status":"UNDEFINED", "msgs":23, "losts":41, "alerts":0,
"alerts_ack":0}, "contrib":{"battery":"ON"},
"type":"event:batteryok", "extra":{}}

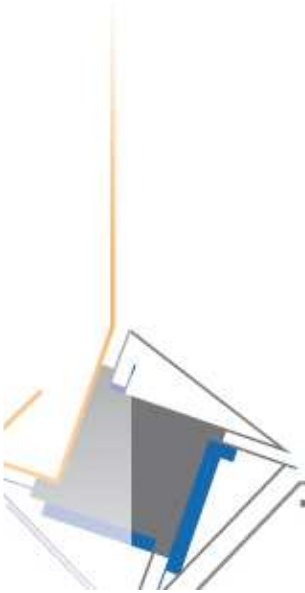
connectingdevices TELECOMDESIGN

De Sensor vers l'infini et au-delà !

Relais par accès API :

Création côté client d'un mécanisme itératif de consultation des messages disponibles sur Sensor.
S'il n'y a pas de nouveaux messages, on se rendort pour une durée définie par le client (ex : toutes les minutes).
S'il y a un nouveau message, on le prend et on le digère (stockage côté client, analyse, représentation graphique, relais, …).

{"when":1365412092833, "lvl":"28.53",
"station":"003B", "ctxt":{"id":"125270139",
"uid":"00001234", "serial":"1FA0/1", "lastseen":"Apr 8,
2013 9:08:12 AM", "firstseen":"Apr 8, 2013 9:03:22
AM", "active":true, "watched":false, "type":"OTHER",
"category":"0000", "index":1, "battery":"ON",
"network":"UNKNOWN", "tamper":"UNKNOWN",
"level":"UNKNOWN", "temp":"UNKNOWN",
"status":"UNDEFINED", "msgs":23, "losts":41, "alerts":0,
"alerts_ack":0}, "contrib":{"battery":"ON"},
"type":"event:batteryok", "extra":{}}



connectingdevices TELECOMDESIGN

De Sensor vers l'infini et au-delà !

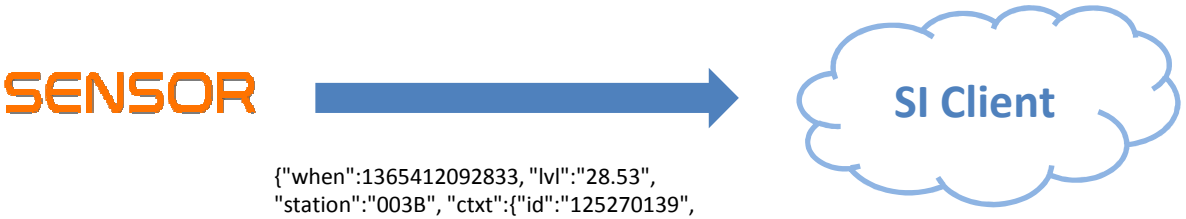
Relais par callback :

Dès qu'un nouveau message est stocké par Sensor, le système relais alors automatiquement un objet *IOT Callback* au SI client à l'url qu'il aura choisie : relais synchrone.

De Sensor vers l'infini et au-delà !

Relais par callback :

Dès qu'un nouveau message est stocké par Sensor, le système relais alors automatiquement un objet *IOT Callback* au SI client à l'url qu'il aura choisie : relais synchrone.

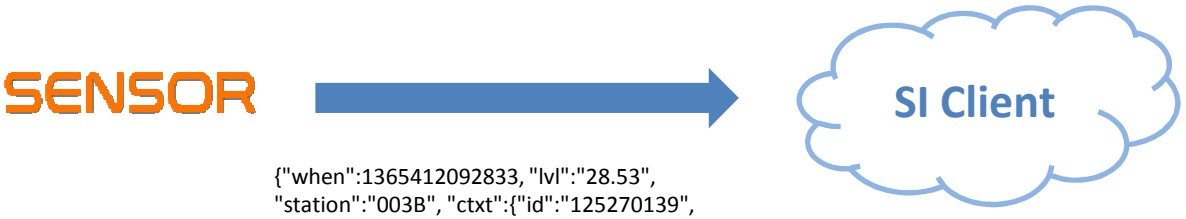


{"when":1365412092833, "lvl":"28.53",
"station":"003B", "ctxt":{"id":"125270139",
"uid":"00001234", "serial":"1FA0/1", "lastseen":"Apr 8,
2013 9:08:12 AM", "firstseen":"Apr 8, 2013 9:03:22
AM", "active":true, "watched":false, "type":"OTHER",
"category":"0000", "index":1, "battery":"ON",
"network":"UNKNOWN", "tamper":"UNKNOWN",
"level":"UNKNOWN", "temp":"UNKNOWN",
"status":"UNDEFINED", "msgs":23, "losts":41, "alerts":0,
"alerts_ack":0}, "contrib":{"battery":"ON"},
"type":"event:batteryok", "extra":{}}

De Sensor vers l'infini et au-delà !

Relais par callback :

Dès qu'un nouveau message est stocké par Sensor, le système relais alors automatiquement un objet *IOT Callback* au SI client à l'url qu'il aura choisie : relais synchrone.



{"when":1365412092833, "lvl":"28.53",
"station":"003B", "ctxt":{"id":"125270139",
"uid":"00001234", "serial":"1FA0/1", "lastseen":"Apr 8,
2013 9:08:12 AM", "firstseen":"Apr 8, 2013 9:03:22
AM", "active":true, "watched":false, "type":"OTHER",
"category":"0000", "index":1, "battery":"ON",
"network":"UNKNOWN", "tamper":"UNKNOWN",
"level":"UNKNOWN", "temp":"UNKNOWN",
"status":"UNDEFINED", "msgs":23, "losts":41, "alerts":0,
"alerts_ack":0}, "contrib":{"battery":"ON"},
"type":"event:batteryok", "extra":{}}

Le relais par callback est privilégié aux accès API pour plusieurs raisons :

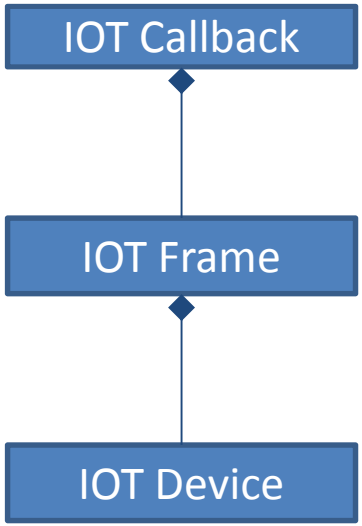
- le serveur relais le message de façon synchrone au client.
- limite les échanges client / serveur au minimum. Seuls des échanges nécessaires sont réalisés.
- pas d'attente active côté client (ou *polling*).

connectingdevices TELECOMDESIGN

De Sensor vers l'infini et au-delà !

Le modèle IOT est utilisé pour relayer un message de Sensor vers un autre SI.

Relayer les trames UDM sérialisées n'a aucun sens, mieux vaut utiliser un modèle objet plus parlant.



Contient des informations sur le relais

Contient des informations sur le message transmis

Contient des informations sur le module communicant

De Sensor vers l'infini et au-delà !

Exemple de callback effectué :


```
{
 "id":"125309902",
 "timestamp":"Apr 10, 2013 1:17:33 PM",
 "msg": {
  "when":1365599849838,
  "lvl":"28.4",
  "station":"003B",
  "ctxt": {
   "id":"125270125",
   "uid":"00001FA0",
   "serial":"1FA0/0",
   "lastseen":"Apr 10, 2013 12:44:23 PM",
   "firstseen":"Apr 8, 2013 9:01:50 AM",
   "active":true,
   "watched":false,
   "type":"OTHER",
   "category":"0000",
   "index":0,
   "battery":"OFF",
   "network":"ON",
   "tamper":"UNKNOWN",
   "level":"UNKNOWN",
   "temp":"UNKNOWN",
   "status":"UNDEFINED",
   "msgs":0,
   "losts":0,
   "alerts":0,
   "alerts_ack":0
  },
  "contrib": {
   "battery":"OFF"
  },
  "type":"event:batterylow",
  "extra": {}
 }
}
```


En rouge : les variables IOT Callback

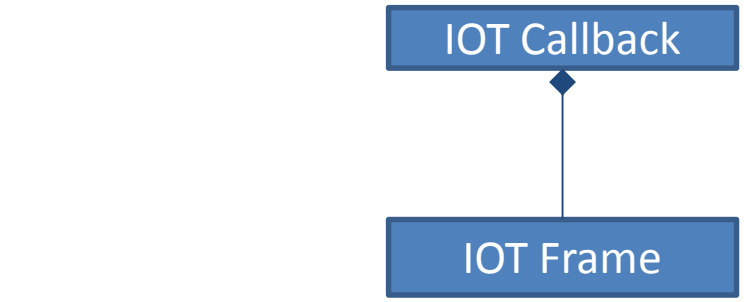
En bleu : les variables IOT Frame

En vert : les variables IOT Device

De Sensor vers l'infini et au-delà !

IOT Callback

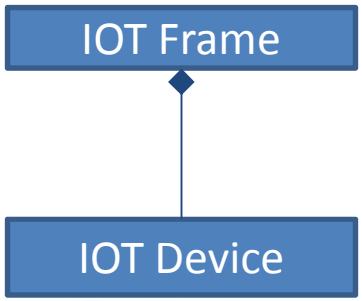
Champs	Type	Description
Id	LONG	Id technique du callback
Timestamp	DATE (GMT)	Date d'émission du callback
Msg	IOT FRAME	Message converti au format IOT Frame
Extra	MAP	Informations additionnelles



De Sensor vers l'infini et au-delà !

IOT Frame

Champs	Type	Description
When	STAMP	Date de réception du message
Lat	STRING	Latitude du module
Lng	STRING	Longitude du module
Level	FLOAT	Niveau de couverture Sigfox du module
Station	STRING	Identifiant de l'antenne Sigfox ayant relayée la trame
Type	STRING	Type de message
Payload	STRING	Détail de la trame
Ctxt	IOT DEVICE	Device concerné
Contrib	MAP	Champs du device impactés par la trame
Extra	MAP	Informations additionnelles



De Sensor vers l'infini et au-delà !

IOT Device

Champs	Type	Description
Id	STRING	Id technique
Uid	STRING	Id du module (version longue de l'id Sigfox)
Serial	STRING	Numéro de série du device
Gateway	STRING	Numéro de série de la passerelle en tête de grappe
Category	STRING (HEXA)	Catégorie du device (dans trame REGISTER)
Index	INTEGER	Position du device (dans trame REGISTER)
Firstseen	DATE	Date d'enregistrement serveur (GMT)
Lastseen	DATE	Dernière date de communication (GMT)
Active	BOOLEAN	Etat technique (ON / OFF)
Status	STRING	Etat fonctionnel
Msgs	INTEGER	Nombre total de messages
Losts	INTEGER	Nombre de messages perdus
Network	ON | OFF | UNKNOWN	Etat du réseau local
Battery	ON | OFF | UNKNOWN	Etat de la batterie
Tamper	ON | OFF | UNKNOWN	Etat du connecteur
Temp	LOW | OK | HIGH	Etat de température
…	…	…

IOT Application

Un objet IOT Application est également disponible pour pouvoir gérer les regroupements de *device*.

Champs	Type	Description
Id	LONG	Id technique de l'application
Created	DATE (GMT)	Date de création de l'application
Name	STRING	Nom de l'application
Description	STRING	Description de l'application
Key	STRING	Clé identifiant l'application
Secret	STRING	Clé secrète utilisée par l'authentification
Callbackurl	STRING	Adresse pour une communication backend/backend
Reftime	DATE (GMT)	Date de référence pour le calcul de variables

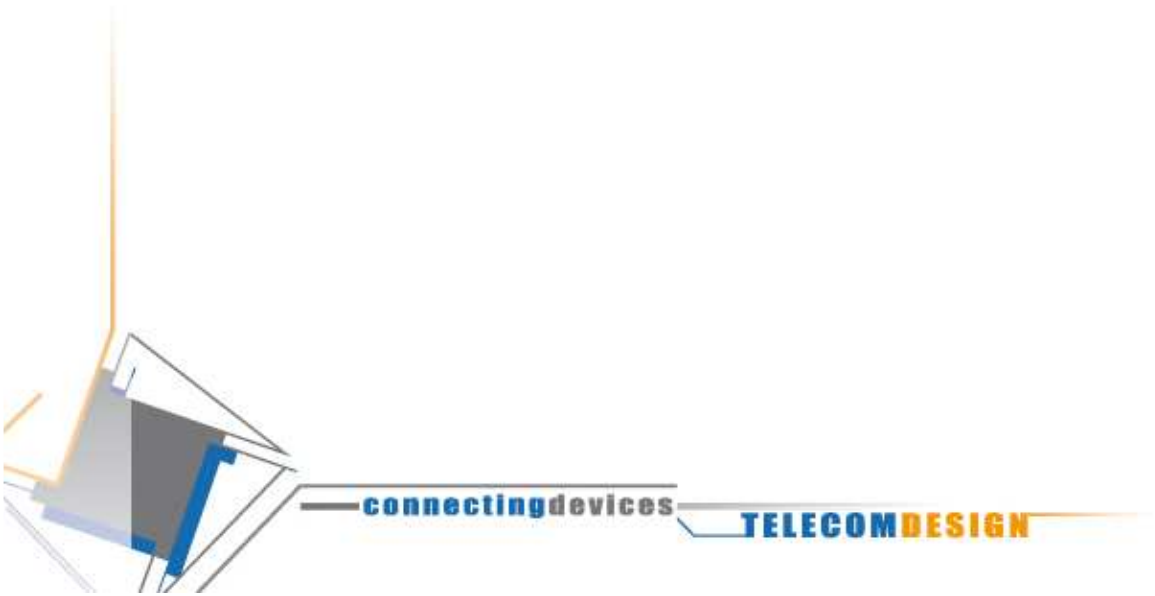
De Sensor vers l'infini et au-delà !

3 niveaux d'API sont disponibles pour interagir avec Sensor :

• **Device API** : permet les opérations de gestion de modules ainsi que la consultation des messages pour un module donné.

• **Developer API** : permet avec une authentification basique HTTP de gérer les applications.

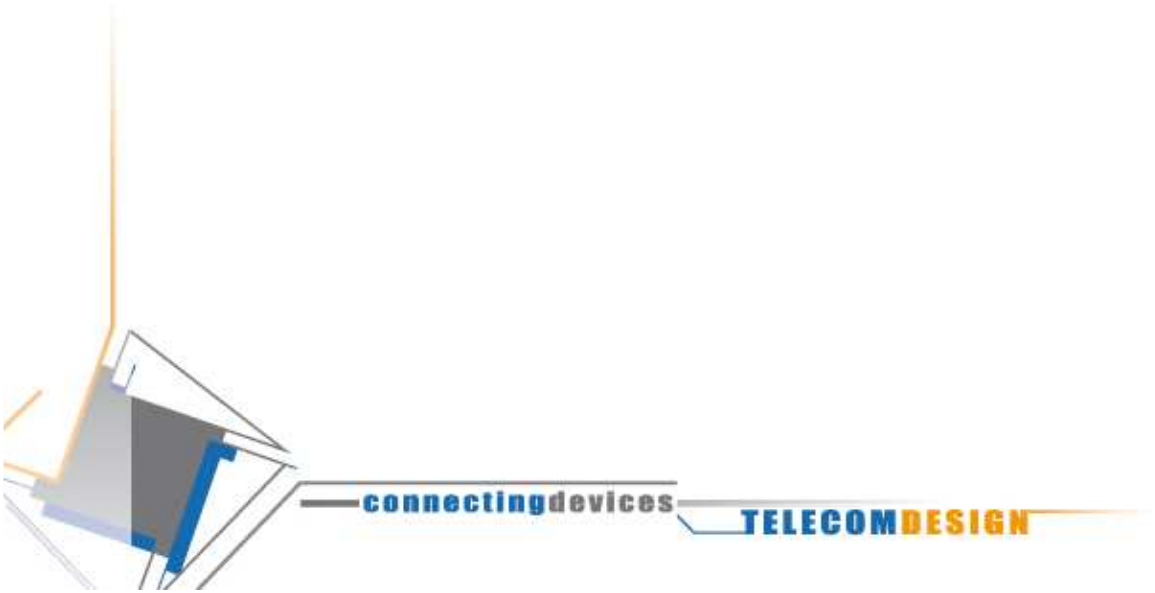
• **Application API** : doit permettre de consulter l'historique des messages et callbacks par application.



De Sensor vers l'infini et au-delà !

Device API

Service		Description	Retour
GET	**/iot/devices/crc.json**	Authentification du device (sn & key)	Un token en base64 à conserver et réutiliser ensuite.
GET	**/iot/devices/msgs/history.json**	Affiche les messages avant telle date (sn & until & amount)	Liste de IOT Frame.
GET	**/iot/devices/msgs/recents.json**	Affiche les messages après telle date (sn & until & amount)	Liste de IOT Frame.
GET	**/iot/devices/children.json**	Retourne la grappe de device derrière une passerelle	Liste de IOT Device.
POST	**/iot/devices/status.json**	Modifie l'état fonctionnel d'un device	Requête de statut 200 si tout s'est bien passé.



De Sensor vers l'infini et au-delà !

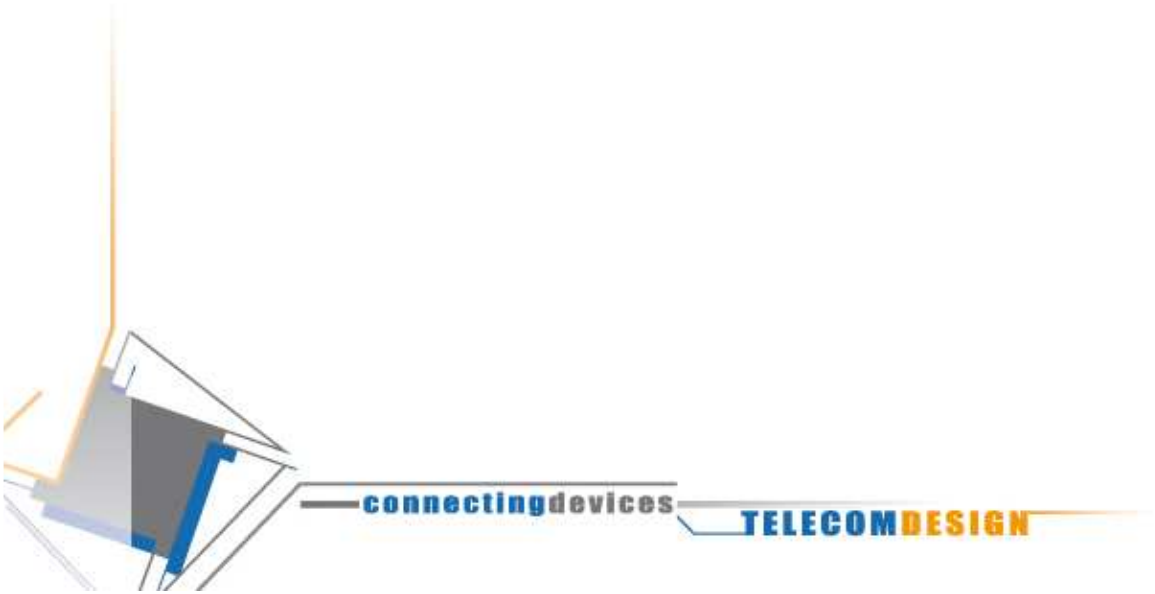
Developper API

Service		Description	Retour
GET | POST	**/security/authentication**	Authentification du développeur	Un token (en base64) à conserver et réutiliser ensuite.
POST	**/iot/developers/register.json**	Enregistrement d'un compte	Un token (en base64) à conserver et réutiliser ensuite.
DELETE	**/iot/developers.json**	Supprime un compte développeur	Requête de statut 200 si tout s'est bien passé.
GET	**/iot/developers/apps.json**	Liste les applications IOT du compte	Liste les applications IOT du développeur.
POST	**/iot/developers/apps.json**	Création d'une application IOT	L'application IOT créée.
DELETE	**/iot/developers/apps.json**	Suppression d'une application IOT	Requête de statut 200 si tout s'est bien passé.
GET	**/iot/developers/modules.json**	Liste les modules rattachés au compte	Liste de modules IOT
POST	**/iot/developers/modules.json**	Rattache un module au compte	Liste de modules rattachés / non trouvés / déjà rattachés / invalides
POST	**/iot/developers/apps/{id}/modules/attach.json**	Attache un module dans une application	Requête de statut 200 si tout s'est bien passé.

De Sensor vers l'infini et au-delà !

Application API

Service	Description	Retour
A DEFINIR	Récupère l'historique des messages d'une IOT Application	**A DEFINIR**
A DEFINIR	Récupère les nouveaux messages d'une IOT Application	**A DEFINIR**
A DEFINIR	Récupère la liste des callbacks effectués pour une IOT Application	**A DEFINIR**



Sensor

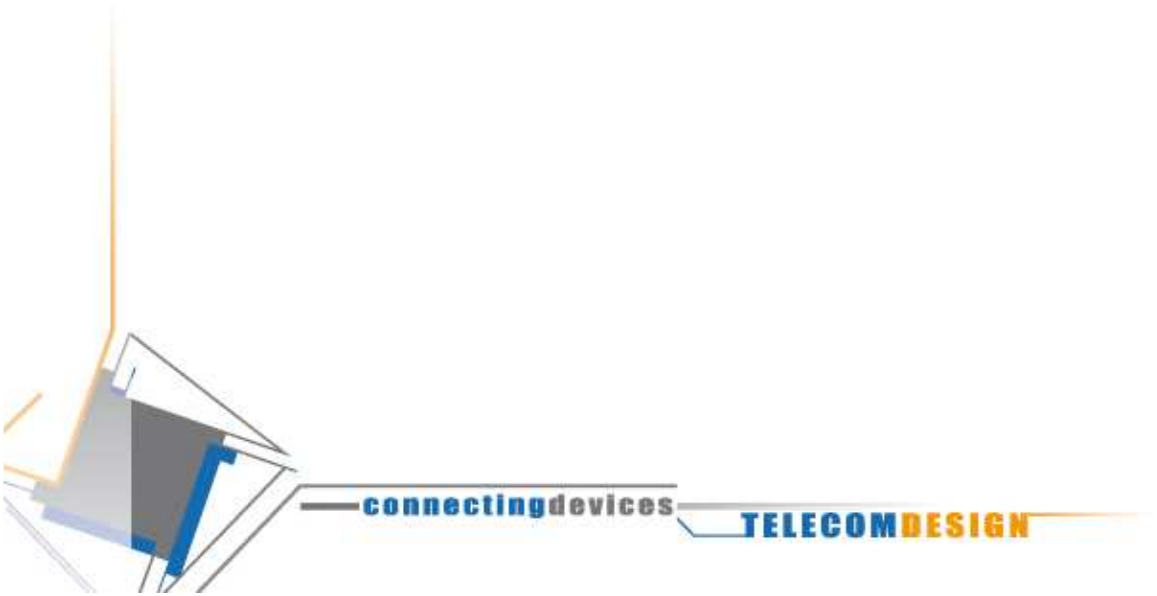
Sensor est le *backend* générique développé par Telecom Design permettant de :

- gérer des appareils communicants
  - ✓ un système est associé à tout appareil communicant permettant de définir des « familles » d'appareils
  - ✓ gérer des états techniques et/ou fonctionnels
  - ✓ effectuer des mises à jour
  - ✓ gérer des grappes d'appareils (une passerelle avec n périphériques)
  - ✓ gérer des groupes d'appareils
- gérer des clients / membres / organisations
- gérer des sites (association entre un appareil et un client à un instant donné)
- gérer des champs *customs* (champs additionnels spécifiques à une entité)
- collecter / stocker / analyser / relayer des données remontées par des appareils communicants
- effectuer des consolidations de données sur la base d'indicateurs existants ou nouveaux

Sensor

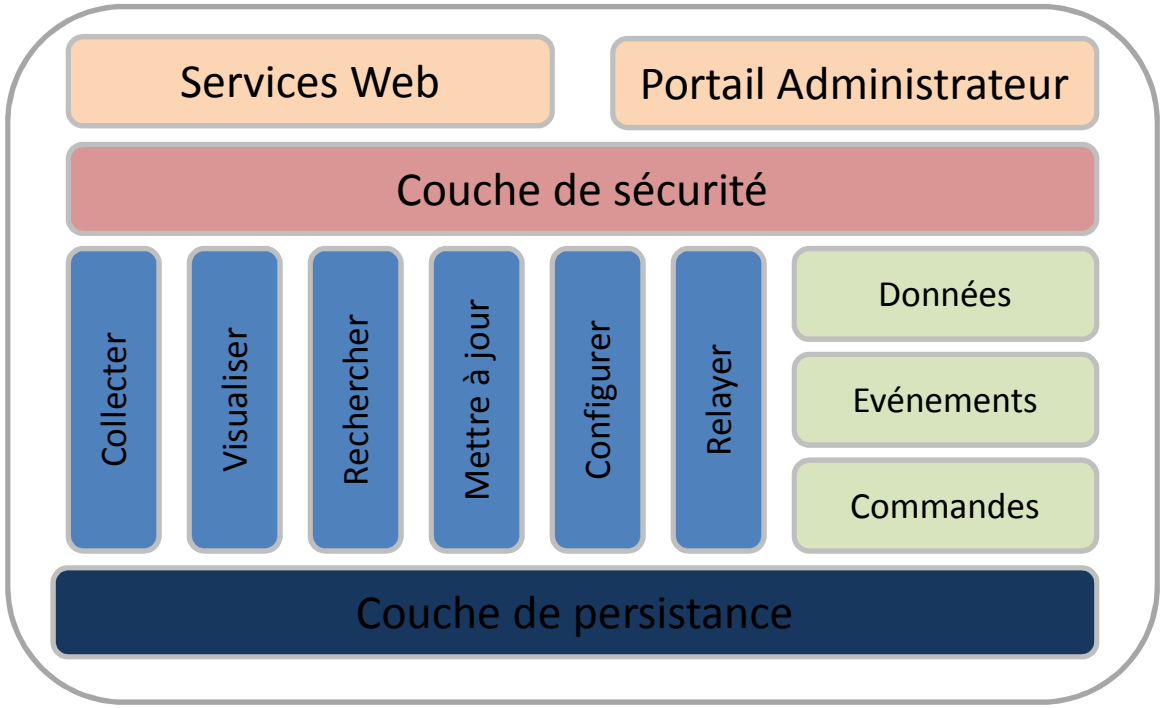
- Visite sur **Sensor** : https://hosting.insgroup.fr

- Visite sur les portails :
  - **Portail solaireControl** : http://solairecontrol.telecomdesign.fr/app/sensor/index.html
  - **Portail MAAF** : https://mcsportal.insgroup.fr (client final & exploitant)
  - **Portail développeur** : https://developers.insgroup.fr



Sensor

Architecture Fonctionnelle



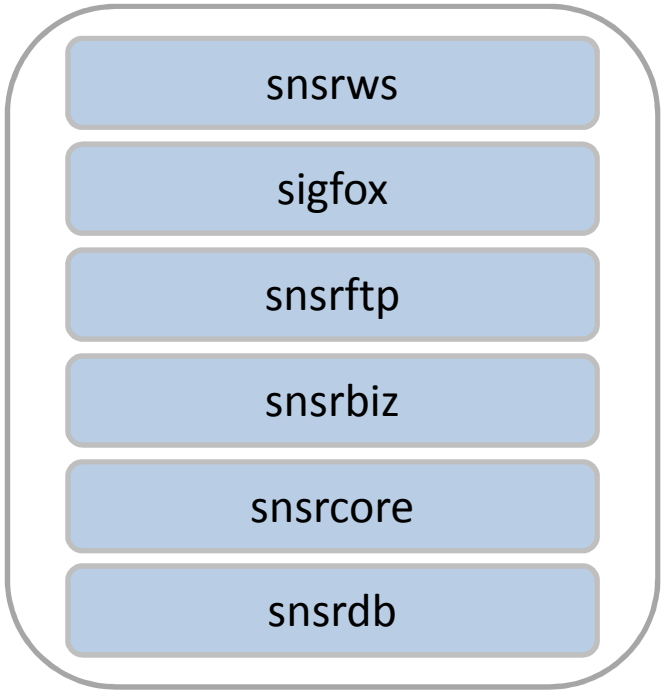
Sensor

Architecture Technique

Sensor est utilise un découpage en modules :
- **snsrdb** : établit le lien entre la couche base de données et le modèle objet associé
- **snsrcore** : couche métier reprenant tous les objets de l'application. Elle porte la logique des règles de gestion sur ces objets
- **snsrbiz** : couche applicative spécifique
- **snsrws** : couche dédiée aux services web

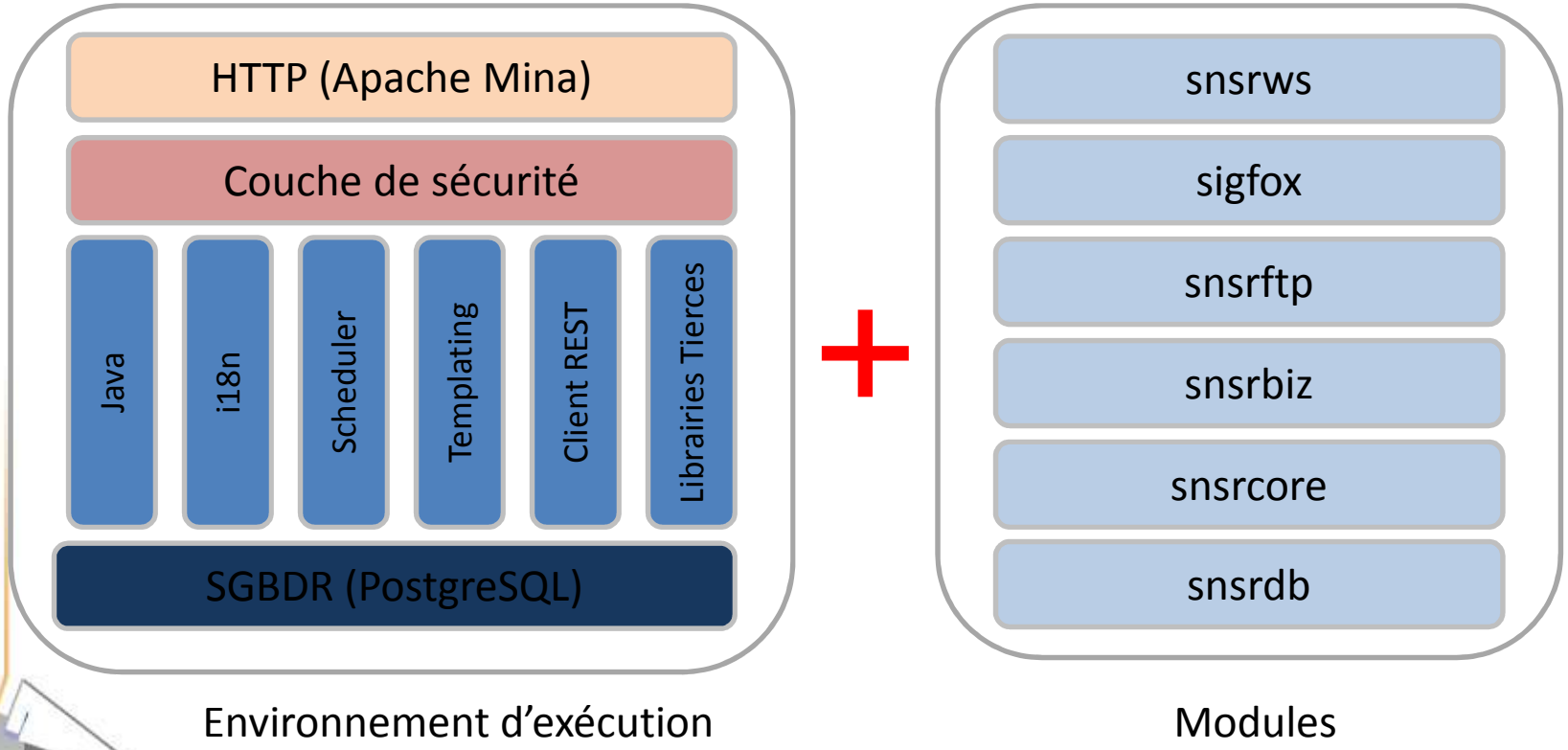
Dépendance entre les modules : **snsrdb** < **snsrcore** < **snsrbiz** < **snsrws**

D'autres modules complémentaires sont disponibles et activables ou pas :
- **snsrftp** : module gérant les accès FTP
- **sigfox** : module permettant des échanges avec Sigfox



Sensor

Architecture Technique : **nœud applicatif**



Environnement d'exécution

Modules

Sensor

Architecture Technique

L'environnement d'exécution ainsi déployé permet de :
- Gérer l'exécution de batchs (*jobs*) avec une programmation type CRON
- Disposer d'un serveur HTTP
- S'affranchir du type de base de données (relationnelle ou pas)
- Utiliser l'internationalisation
- Fournir des services Web REST (développement de l'architecture orienté ressources)
    - les services REST sont sans état (chaque requête envoyée vers le serveur doit contenir toutes les informations à leur traitement)
    - minimisation des ressources systèmes, pas de session ni d'état
    - les services REST fournissent une interface uniforme basée sur les méthodes HTTP :
        - ✓ Création par : POST
        - ✓ Lecture par : GET
        - ✓ Modification par : PUT
        - ✓ Suppression par : DELETE
    - les architectures REST sont construites autour de ressources qui sont identifiées par des URI
    - la représentation des données se fait en JSON (moins verbeux que XML).

Sensor

Architecture Technique

Cette solution est :

- clonable (car sans état)
- configurable
- portable (Linux, Windows)
- stable (basée sur Java)

Elle autorise :

- La scalabilité horizontale : possibilité d'ajouter des serveurs avec répartiteur de charge.
- La scalabilité verticale : possibilité d'upgrader un serveur (ajout de processeurs, RAM, disques…), y compris, sur demande.

TP

// DEVICE API
- Accéder à l'historique des 20 derniers messages émis par mon module (soit par *curl*, java ou autre)

// DEVELOPER API
- Enregistrement d'un nouveau compte développeur

- Utilisation de l'API pour s'authentifier

- Lister les applications du développeur

- Ajouter / Supprimer une application

- Déployer l'application sur *cloudbees*

- Tester le mode callback (définir une IOT Application pour mon module, puis faire un *controller* qui trace l'arrivée d'un *callback* sur le portail *cloudbees*)

- Envoyer un *email* si le portail déployé sur *cloudbees* reçoit un événement batterie faible

- Présentation de IFTTT pour poster un tweet sur un événement batterie faible

Annexe 1 : Device API

Authentification pour le module 1FA0 avec sa clé XXXXXXXX : permet de récupérer le token nécessaire à l'utilisation des autres services de l'API
> curl -X GET "https://sensor.insgroup.fr/iot/devices/crc.json?sn=1FA0&key=XXXXXXXX"
>>> NDAwMDFGTVDMjczREE=

Récupérer le dernier message émis par le module 1FA0 :
> curl -X GET -H "X-Snsr-Device-Key: **NDAwMDFGTVDMjczREE=**" "https://sensor.insgroup.fr/iot/devices/msgs/history.json?sn=1FA0&amount=1"
>>> [{"when":136577015788,"lvl":"30.56","station":"003B","ctxt":{"id":"12512","uid":"00001FA0","serial":"1FA0/0","lastseen":"Apr 12, 2013 12:35:57 PM","firstseen":"Apr 8, 2013 9:01:50 AM","active":true,"watched":false,"type":"OTHER","category":"0000","index":0,"battery":"OFF", "network":"ON","tamper":"UNKNOWN","level":"UNKNOWN", "temp":"UNKNOWN","status":"UNDEFINED","msgs":0,"losts":0,"alerts":0,"alerts_ack":0},"contrib":{"battery":"OFF"},"type":"event:batterylow","extra":{}}]

Récupérer les 20 derniers messages (valeur par défaut) émis par le module 1FA0 :
> curl -X GET -H "X-Snsr-Device-Key: **NDAwMDFGTVDMjczREE=**" "https://sensor.insgroup.fr/iot/devices/msgs/history.json?sn=1FA0"

Récupérer les 20 derniers messages émis avant le vendredi 12 avril 2013 14:33:00 par le module 1FA0 :
> curl -X GET -H "X-Snsr-Device-Key: **NDAwMDFGTVDMjczREE=**" "https://sensor.insgroup.fr/iot/devices/msgs/history.json?sn=1FA0&until=1365769980000"

Récupérer les 5 derniers messages émis après le 2013-04-10 14:44:23 par le module 1FA0 :
> curl -X GET -H "X-Snsr-Device-Key: **NDAwMDFGTVDMjczREE=**" "https://sensor.insgroup.fr/iot/devices/msgs/recents.json?sn=1FA0&amount=5&after=1365544800000"

Lister la grappe de devices rattachée au module 1FA0 :
> curl -X GET -H "X-Snsr-Device-Key: **NDAwMDFGTVDMjczREE=**" "https://sensor.insgroup.fr/iot/devices/children.json?sn=1FA0"

Changer l'état fonctionnel du module 1FA0 (son id technique étant 8255) en "TEST" :
> curl -X POST -H "X-Snsr-Device-Key: **NDAwMDFGTVDMjczREE=**" "https://sensor.insgroup.fr/iot/devices/status.json?id=8255&value=TEST" -d '{}'

Annexe 2 : Developer API

Prérequis : avoir un compte développeur créé depuis https://developers.insgroup.fr/register.html
Cette étape permet d'obtenir un login & password par développeur. Par exemple login@host.fr (login) & 3fe50627c371094b8ed3c6 (password)

Authentification pour le développeur login@host.fr (mot de passe 3fe50627c371094b8ed3c6) : permet de récupérer le token nécessaire à l'utilisation des autres services de l'API
> curl -X GET "https://sensor.insgroup.fr/security/authentication?login=login@host.fr&pwd=3fe50627c371094b8ed3c6"
>>> Y2hyaXN0b3VpbGhlQGMzFkYTQ2Yzg5MzE=

Créer une application s'appelant myApp et dont l'url de callbaclk est http://my.callback.url :
> curl -X POST -H "Authorization: Basic **Y2hyaXN0b3VpbGhlQGMzFkYTQ2Yzg5MzE=**" https://sensor.insgroup.fr/iot/developers/apps.json -d '{"name": "myApp", "description" : "My app description", "callbackurl" : "http://my.callback.url"}'

Lister mes applications :
> curl -X GET -H "Authorization: Basic **Y2hyaXN0b3VpbGhlQGMzFkYTQ2Yzg5MzE=**" https://sensor.insgroup.fr/iot/developers/apps.json

Lister mes modules enregistrés :
> curl -X GET -H "Authorization: Basic **Y2hyaXN0b3VpbGhlQGMzFkYTQ2Yzg5MzE=**" https://sensor.insgroup.fr/iot/developers/modules.json

Déclarer la possession d'un nouveau device 1FA0 (dont la clé est XXXXXXXX) pour mon compte développeur :
> curl -X POST -H "Authorization: Basic **Y2hyaXN0b3VpbGhlQGMzFkYTQ2Yzg5MzE=**" "https://sensor.insgroup.fr/iot/developers/modules.json" -d '[{"serial": "1FA0", "key": "XXXXXXXX"}]'
>>> {"registered":["1FA0"],"not_found":[],"already_registered":[],"invalid_key":[]}

Lister mes modules enregistrés :
> curl -X GET -H "Authorization: Basic **Y2hyaXN0b3VpbGhlQGMzFkYTQ2Yzg5MzE=**" https://sensor.insgroup.fr/iot/developers/modules.json
>>> [{"id":"9226","serial":"1FA0"}]

Enregistrer un nouveau device 1FA0 (dont on a déjà déclaré être possesseur) à mon appli (dont l'id technique est 52664) :
> curl -X POST -H "Authorization: Basic **Y2hyaXN0b3VpbGhlQGMzFkYTQ2Yzg5MzE=**" "https://sensor.insgroup.fr/iot/developers/apps/52664/modules/attach.json" -d '{"serials": "1FA0"}'

Supprimer une application (dont l'id technique est 52664) :
> curl -X DELETE -H "Authorization: Basic **Y2hyaXN0b3VpbGhlQGMzFkYTQ2Yzg5MzE=**" "https://sensor.insgroup.fr/iot/developers/apps.json?id=52664" -d '{}'