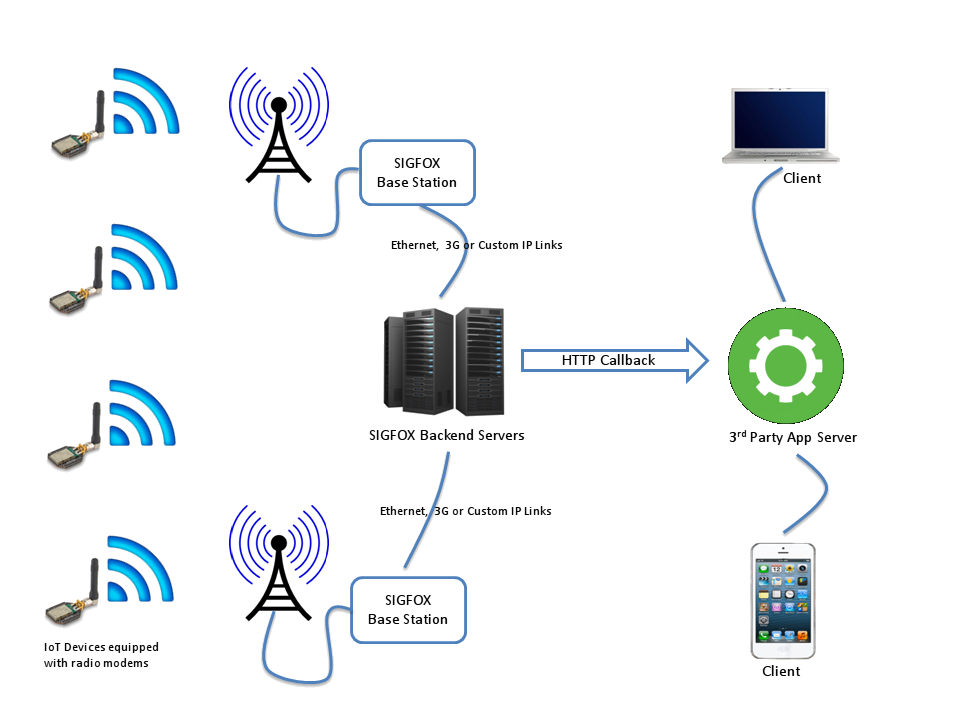
Intro to the Sigfox

SIGFOX is a connectivity solution that focuses on low throughput devices. On SIGFOX your device can send between 0 and 140 messages per day. This is a maximum of one message every 11 minutes, if you want a 24h distribution. And **each message can be up to 12 bytes of actual payload data**. You can also transmit 4 messages of 8 bytes payload to each device per day. In order to receive messages, the device has to request for data from your server, which means that it has to be programmed to receive data at specific events or at specific times.



There are two ways to get data from sigfox:

* Callback URL from SIGFOX to TW
* Direct call using the SIGFOX REST API from TW

Callback API  
Each time a sensor is sending a message over the SigFox network this one is recorded in the sigfox backend. You can configure a call back in this backend to indicate what action you want to perform with this message. You can also configure a callback that sends multiple messages from the servers to your application in a single request.  This option is particularly interesting if you have a lots of sensors because you won’t spam Thingworx with messages for each message received. Sigfox will couple multiple messages together and only call your service once.

The nice thing about this callbacks and thingworx, is that Sigfox can send the data received from the device as a parameter to a Thingworx service, using the TWX Rest API.

For example, if you have a thing called **SigfoxInterface**, with a service **CallbackDataBatch,**with a string parameter, this service can be called by sigfox using  a callback configured as following:

URL pattern:  [https://THINGWORX\_IP/Thingworx/Things/SigfoxInterface/Services/CallbackDataBatch?appKey=APP\_KEY&batch={batch}](https://thingworx_ip/Thingworx/Things/SigfoxInterface/Services/CallbackDataBatch?appKey=APP_KEY&batch=%7bbatch%7d)

Line Pattern:  {time};{device};{snr};{rssi};{avgSignal};{station};{data};{lat};{lng};{seqNumber};

REST API

Sigfox also provides an extensive rest API. Some of the features found on the SIGFOX backend’s website can also be accessed programmatically using a webservice API. This API uses the HTTP protocol, following the REST principles.

All API endpoints return data in the JSON format, with the corresponding «application/json» content type header.

You can use the API to create new device types or devices, new callbacks or get data for devices.

One of this REST services is [https://backend.sigfox.com/api/devices/**DEVICE\_ID**/messages?limit=1](https://backend.sigfox.com/api/devices/DEVICE_ID/messages?limit=1), that returns the total number of device messages for one device, this day, this week and this month.

Sigfox Application template

Because much of the first steps of creating a Sigfox project are the same for each use case, i've prepared a small application template. This template is by no means complete. You will most likely need to edit it to fit your application.

Inside this application you can find the following entities:

* **​Sigfox.GenericDeviceThingShape**: ​​​A generic thingshape with properties that you can apply on all your sigfox enabled things. This thing shape encapsulates properties like the timestamp of the last messages, the location of the device, state, sequence numbers, and much more. Some of the properties are logged, so be carefull because you'll need to provide your own Value Stream. The thingshape also wraps some alerts: for example, messages received out of order and for lack of communication.
* **Sigfox.DevicesConnectedScheduler:**Used to raise the alert that a ​Sigfox.GenericDeviceThingShape has not received messages.
* **Sigfox.RestInterface:** Maps a couple the most used REST API endpoints: will help with device type and device listing, callback creation, enrollment of new devices. Consult the services list for more info. To configure, just use you API credentials in the properties.
* **Sigfox.CallbackInterface:** Contains the service that is called by SIGFOX as a callback. you can see the example above for how to configure the on the Sigfox side. The service **CallbackDataBatch**will match the device in the callback to a Thing with the ​**Sigfox.GenericDeviceThingShape**, update its properties, and then call the **UpdateData**service, with the parameters **data**(the raw data from the device) and **receiveTimestamp**(when Sigfox says it received the data). The implementation of this service is application specific, so it is not included in this package. Its job is to parse the raw data and set device specific properties.

[[SigfoxTemplate.xml](https://share.ptc.com/sites/sales/ic/IoT%20Presales%20Enablement/blog/SiteAssets/Lists/Posts/NewPost/SigfoxTemplate.xml)SigfoxTemplate.xml](https://share.ptc.com/sites/sales/ic/IoT%20Presales%20Enablement/blog/SiteAssets/Lists/Posts/NewPost/SigfoxTemplate.xml)

Frame Utilities Extension

Decoding and encoding raw binary data is an annoying task. Bit shifting and masking takes a lot of time and debugging, and, with sigfox, there is no way to get around this. The Frame Utilities Extension greatly simplifies this: all you need to do is create an infotable using the **FrameFormatDataShape**that maps the frame definitions received from the device manufacturer.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ​**BITS** | **0-7** | **8** | **9-15** | **16-23** | **24-39** | **64-79** |
| ​**FIELD** | CODE | Conf Status | Frame counter | Transmit Period | Sensor Value (LSB) | Sensor Offset  (LSB) |

The extension provides a way to parse the encode and decode the data. For more details refer to the attached documentation.

[[MessageUtilitiesExtension.zip](https://share.ptc.com/sites/sales/ic/IoT%20Presales%20Enablement/blog/SiteAssets/Lists/Posts/NewPost/MessageUtilitiesExtension.zip)MessageUtilitiesExtension.zip](https://share.ptc.com/sites/sales/ic/IoT%20Presales%20Enablement/blog/SiteAssets/Lists/Posts/NewPost/MessageUtilitiesExtension.zip)

P.S. There is no warranty on this extension. It was tested on restricted data sets, it should work, but please test yourself on some sample data. Also, if you make and changes, since the extension is in javascript, share them in the comments.