Exosite Solutions Guide Documentation

Release 1

Andy Lee

Aug 01, 2017

Contents

This guide is intended to give the user an introduction to building an IoT solution based on the Exosite cloud. This guide encompasses getting data from your physical devices up to the Exosite OnePlatform.

Exosite also has an optional frontend component available. The frontend product is called Portals. Further information about Portals can be read at a yet to be determined place.

If you are new to Exosite, please see our exosite_introduction document. It will get you familiar with terms and concepts used in describing/designing your solution. Once you are familiar with Exosite please checkout the rest of our guides

Guides:

CHAPTER 1

Provisioning

For a physical device to communicate with with Exosite, it must first obtain a CIK. Before this can happen, you need to add the device to your Exosite account using Portals. The CIK can than be obtained one of two ways:

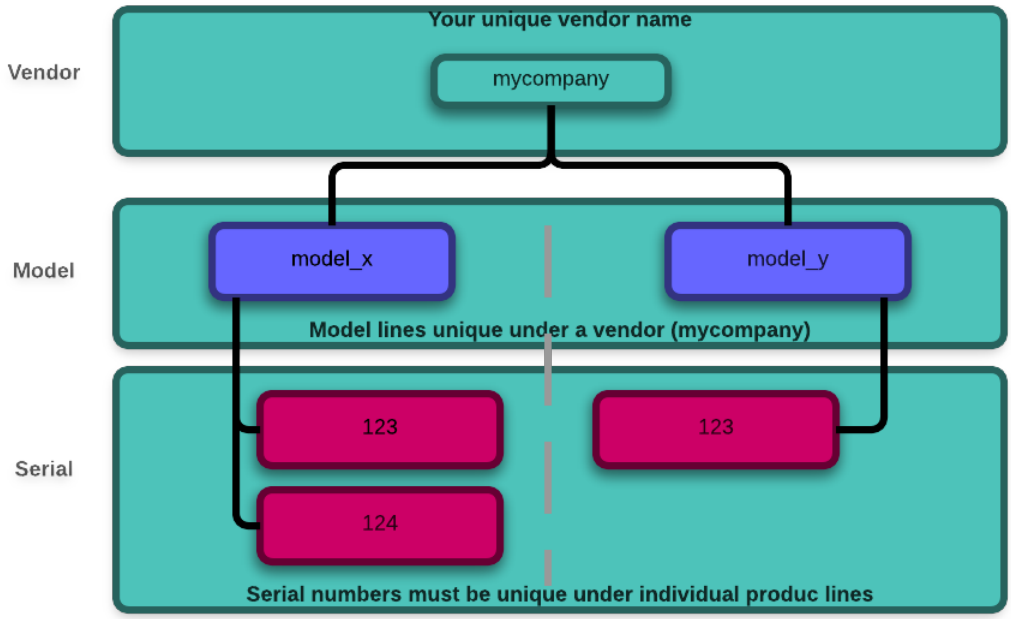
1. Hardcoding the CIK into your application code
2. Retrieving the CIK over the provisioning API.

This document covers how to retrieve the CIK through the provisioning API.

The Exosite provisioning API requires 3 pieces of information:

1. `vendor`
2. `model`
3. `sn` (serial number)

The vendor and model are set when the client model is created. The *vendor* name is unique among all Exosite customers, the *model* is unique under each Exosite customer, and the *sn* is unique among all models.



We will be covering the portion of the API that allows a device to retrieve its CIK, but more detailed documentation about the provisioning API can be found on Github

A device can be in 3 different states.

1. *Pending Activation*
2. *Activated*
3. *Expired*

When a device is first added to a Portal, it is in the *Pending Activation* state. This means that a device is eligible to activate on the platform, but has not yet done so. If a device doesn't successfully activate after 24 hours, it will go into the *Expired*. This means that even if the device calls in to activate it will not succeed. The device will nee to be "re-enabled" before it can successfully complete its activation. If a device does successfully activate within the 24 hour window. It will be in the *Active* state and able to receive data through the OneP API's.

The provisioning API allows a device to retrieve its CIK. This method only works when a device is unactivated. After the call has been made the device will activate and it will no longer be able to retrieve its CIK through this API.

The HTTP request to retrieve a CIK looks like the following:

```
POST /provision/activate HTTP/1.1
Host: m2.exosite.com
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Content-Length: <length>

vendor=<vendor>&model=<model>&sn=<sn>
```

Where <*vendor*>, <*model*>, and <*sn*> will be specific for your individual project.

This request can return four different response codes:

1. *200* – Request was accepted and the response body will have the CIK
2. *404* – The serial number of the device isn't in the 1P
3. *403* – Invalid vendor/model pair

4. *409* – Device has not yet been added to a portal, or it has already been added to a portal and gone through its activation process.

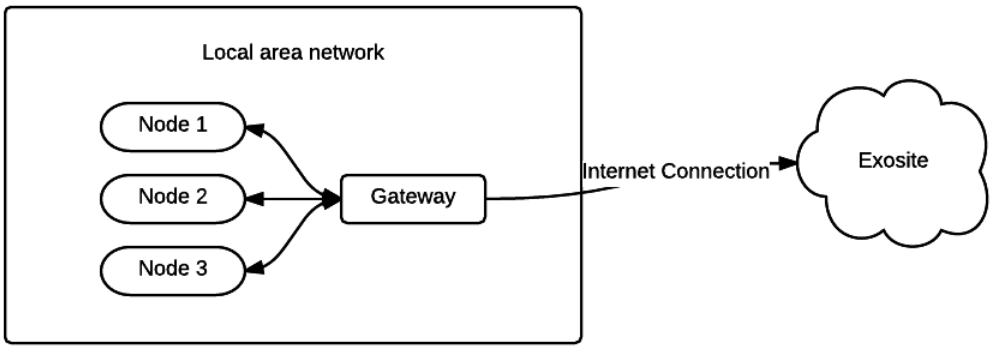
CHAPTER 2

Gateway Design Guide

This document is intended to describe some typical uses of a gateway/node system and design guidelines for implementing a system.

What Is A Gateway/Node System?

Gateway/node systems are described by a system where one, or many, node devices uses the Internet connection of a single device (the gateway). The gateway typically communicate with the nodes via a local wire or wired network (e.g. BACnet, Bluetooth, CAN, Modbus...) and communicates with Exosite via a cellular, Ethernet, or WiFi network.



Gateway A device that has a connection to the Internet. This connection can be WiFi, Ethernet, cellular, or any other link that allows it to talk to Exosite. The gateway will also have a way to communicate with local nodes

Node A device that has data it needs to send to Exosite, but it doesn't have its own connection to Exosite. It therefore relays that data through a gateway.

Gateway/Node System Architectures

There are typically two different ways that the gateway/node system send data up to Exosite:

1. Batch sending
2. Real-time sending

Batch sending

With the batch sending method, when the gateway receives data from the nodes, it will store the data in memory, or on disk. At regular intervals (e.g. once a day) it will send all of the stored data up to Exosite.

Advantages

- Since the gateway opens a socket to Exosite at a slower rate (e.g. once per day) the system is able to reduce the data overhead of opening a socket for every piece of data
- For low power systems, the modem wakes up less often, thus improving battery life.

Disadvantages

- The main disadvantage to the batch sending method is the loss of real-time data viewing. By storing the data on the local gateway, the end user is unable to see that data until the gateway sends the data up to Exosite.
- The end user is unable to receive real-time alert notifications (e.g. over temperature condition).

When using the batch send method, it typically also requires that your gateway have access to an accurate time source. This allows the gateway to store the actual timestamp of when the node sends the data. If you have an accurate timestamp, when all of the batch data is finally sent to Exosite, the timestamps can be included with it, allowing the UI to be able to show when each individual datapoint occurred.

Real-time sending

With Real-time sending, the gateway sends data as soon as it receives it from the nodes.

Advantages

- The user is able to always see the most recent data in their cloud UI
- Notifications are able to be sent immediately when a threshold is crossed.
- Since Exosite is able to timestamp the record as soon as it receives it, the gateway does not necessarily need access to a time source.

Disadvantages

- The data overhead of all these connections uses much more data than a single batch report.
- If an Internet connection is unavailable, the application will either need to throw the data on the ground, or decide how to buffer it.

The overhead of a typical TCP/IP/HTTP request/response is around 1.3kB. This means that even though your payload may only be 20 bytes long, each request/response will use 1.3kB of data. On cellular networks this has the potential to add up quickly and result in unexpected cellular data usage numbers.

Hybrid approach

Often times a solution ends up being a hybrid of the above two approaches. Using a hybrid approach allows the application to gain some of the benefits of each approach while still keeping some of the benefits of the other approach.

One popular method is to batch data and send on regular intervals, but if some predefined condition happens the batched data can be sent up earlier. For example a gateway is programmed to batch a nodes temperature data for one day. At the end of a 24 hour period, the gateway should send all of the batched data for the previous day.

Let's say, for this example, that the user wants to be notified when the temperature goes above 40. If the application was developed using a pure batch and send method, the user wouldn't know that the temperature violation occurred until after the 24 hour report had been sent. If we instead update the gateway app to send its batched data every 24 hours, or when it receives a reading over 40, we save on bandwidth usage during normal operations but are still able to get real-time notifications of when violations occur.

In the above situation the gateway acts in batch mode when things are working as expecting and goes to real-time mode when a special condition happens.

Another potential architecture is to have the device send all data up in real time, but when the connection to Exosite goes down, fall back to a batch mode to keep data points stored locally on the gateway until the connection is restored and the points can be sent to Exosite. This allows you to keep all the benefits of real-time reporting, but you are also able to handle periods of offline time.

Important consideration when choosing an architecture

Often times the deciding factor for choosing an architecture is how the gateway will connect to Exosite. If the connection is over cellular, data usage tends to be much more expensive and a more batch send type architecture is used. If the connection is using WiFi or Ethernet, data usage typically isn't a concern and the architecture slides more towards the real-time end.

Choosing a connection type

Connection types are typically divided into the follow three categories:

1. Cellular
2. WiFi
3. Ethernet

Cellular

Cellular devices use a cellular network to communicate with One Platform. The networks used for communications are often the same networks that cell phones use.

WiFi

Gateways that use WiFi use the same WiFi networks that your computer connects to.

Ethernet

Ethernet connections are hardwired connections that connect directly to a network with access to the

Given the choice, a non-cellular connection type is almost always the preferred method of connection. However, there is one potential drawback to using the end user's Ethernet connection, and that is their network infrastructure. Often times an end users network will have firewalls and/or proxies in place to protect against malicious activity. Unfortunately, these security devices also can hamper your gateway's activity. Making it difficult, or impossible for your data to reach Exosite's servers.

When choosing the gateway's Internet connection, it often involves a balance between the complexities of navigating the end users IT network policies and the cost of a recurring cellular bill, in combination with the desired architecture (batch vs real-time sending).

General best practices

- When possible, always UTC time or Linux epoch time. This makes dealing with different timezones much easier
- Use separate threads for asynchronous communications to the local network and Exosite

Additional Topics

Choosing a Device Hierarchy

When designing a solution that utilizes one, or many, gateways with nodes attached to that gateway, a decision needs to be made about how to handle the CIKs for each device. This page will give an introduction to the topic of choosing how to store these ciks

Common Hierarchies

How to structure the hierarcy of the devices in your system typically falls into one of three different ways.

1. Each node and gateway stores and uses its own cik
2. Each node and gateway has its own cik, but the gateway stores the cik for each node
3. Only the gateway has a cik and it uses

Which Hierarchy To Choose?

Choosing one of the three methods is a decision that should be made after you know what type of hardware you will be using, and what type of information you want to send to One Platform.

Options #1 and #2 both use the same concept of one cik per device, whereas option #3 uses one CIK per node/gateway group. The most flexible solution is a one-to-one relationship between physical devices (nodes/gateway) and the devices on your platform. This allow you to add/remove devices as nodes are updated/replaced, without having to change your data model.

There are, however, times when you will only want one cik per gateway and all of it's nodes. This may happen when the configuration of nodes/gateways is always the same. For example, your system measures tire pressures on a motorcycle. In this instance you know that you will always have a front tire pressure and a rear tire pressure sensor,

and one gateway. In this case, even if you swap out sensors, you still have one front pressure sensor and one rear pressure sensor.

The only difference between options #1 and #2 is the where the CIK is stored. Often times your sensor node devices may not be able to store their cik on board or they are already in working systems and you do not want to alter their firmware/software. If this is the case you will want to go with option #2. This adds more complexity and additional state to the gateway.

Alternatively this complexity and state can be moved to the sensor nodes. The main advantages to storing the cik on the nodes is the ability to decouple the node devices from the gateway devices.

Other advantages?

API Usage

This is meant to be a high level primer on using Exosite's APIs. For the full API documentation, please see the official Exosite docs page (http://docs.exosite.com/). Before reading this document, you should be familiar with HTTP. If you're not familiar, this page gives a good introduction.

This page does not cover the device provisioning process for details on the API's required to provision your device, please see the *Provisioning* doc.

Data, RPC, or CoAP?

Exosite API calls can be made using the Exosite Data Interface, the Exosite JSON RPC interface, or the Exosite CoAP Interface. Which API you choose depends on the architeture of your gateway.

Data API

The Data API is a simple API built on top of HTTP. It uses HTTP and provides the following basic functions.

- Write
- Read
- Read/Write
- Long-Polling

Write

If you want to write the value `24` to your devices `temperature` datasource, the body of your `POST` request is simply `temperature=24`.

Data API Example:

```
POST /onep:v1/stack/alias HTTP/1.1
Host: m2.exosite.com
X-Exosite-CIK: < your cik >
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Content-Length: 14

temperature=24
```

Read

If you want to read from a datasource, you make a `GET` request to `GET /onep:v1/stack/alias?<alias 1>` where `<alias 1>` is the name of your devices datasource you want to read from. For example, if you want to read from your device's `temperature` datasource, you would make a `GET` request to `GET /onep:v1/stack/alias?temperature`, and the body of the response would look like this: `temperature=24`.

Data API Example:

```
GET /onep:v1/stack/alias?temperature HTTP/1.1
Host: m2.exosite.com
X-Exosite-CIK: < your cik >
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Content-Length: 0
```

Note: Your read responses will have the body url encoded

Note: With the data API, if your requests has multiple aliases, they will be seperated by a `\r\n` in the body.

If you want to read from multiple datasources at the same time, you would add multiple datasources as URL parameters and seperate them with an `&` (e.g. `/onep:v1/stack/alias?temperature&humidity`). The response would contain your read values seperated by an `&` (e.g. temperature=25&humidity=77).

Read/Write

The read/write command allows you to perform a read and a write request at the same time. To do this, you combine both of the above methods and make a `POST` request to `/onep:v1/stack/alias?<alias 1>` while also including your write values in the body of your request. The response of your request will contain the values of your read requests.

Data API Example:

```
POST /onep:v1/stack/alias?temperature HTTP/1.1
Host: m2.exosite.com
X-Exosite-CIK: < your cik >
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Content-Length: 14

temperature=24
```

JSON RPC

The JSON RPC uses commands encoded in as JSON. Using the JSON RPC, you can send multiple commands at once. It provides the most features of any of Exosite's APIs.

The main disadvantage to using the JSON RPC is that it requires the most bandwidth and it also requires the application to parse/build json. Most higher level languages have support for this. If you are developing in C, Exosite has successfully used the Jsmn library for parsing JSON. For more details on using Jsmn in your project, please see the **'Jsmn development guide <>'_**.

Here is a full list of the JSON RPC commands. We will cover a small subset of the commands that allow your device to read/write data to Exosite.

CoAP

The CoAP API is intended to be used for low bandwidth devices.

DTLS

The CoAP API also has the ability to use a form of DTLS to keep the link between Exosite and your device private.

Gateway Engine

Gateway Engine is a framework that eases the development of gateway applications. This page gives a brief introduction to what te Exosite Gateway Engine is. For more detailed documentation, please see the project's readme file.

Key Features

It provides the following key features:

Application Hosting

Gateway Engine's core feature is to provide a framework for your application/s to run in. This allows you to focus development efforts on just the business logic of your application.

Process Monitoring

Gateway Engine incorporates Supervisord to watch application processes and make sure that if they die, they will be restarted.

Application Logging

Gateway Engine captures all of an applications stdout and writes that to a log file. The Log files are automatically rotated and the oldest ones deleted, ensuring that your logs won't fill up your disk space.

Bandwidth Usage and Monitoring (Beta)

If using the Exo-Python module, Gateway Engine can automatically watch your bandwidth for you and help make sure that you don't have any unexpected costly cellular bills.

> **Warning:** This feature is still in early beta and is still being tested.

Application Updates

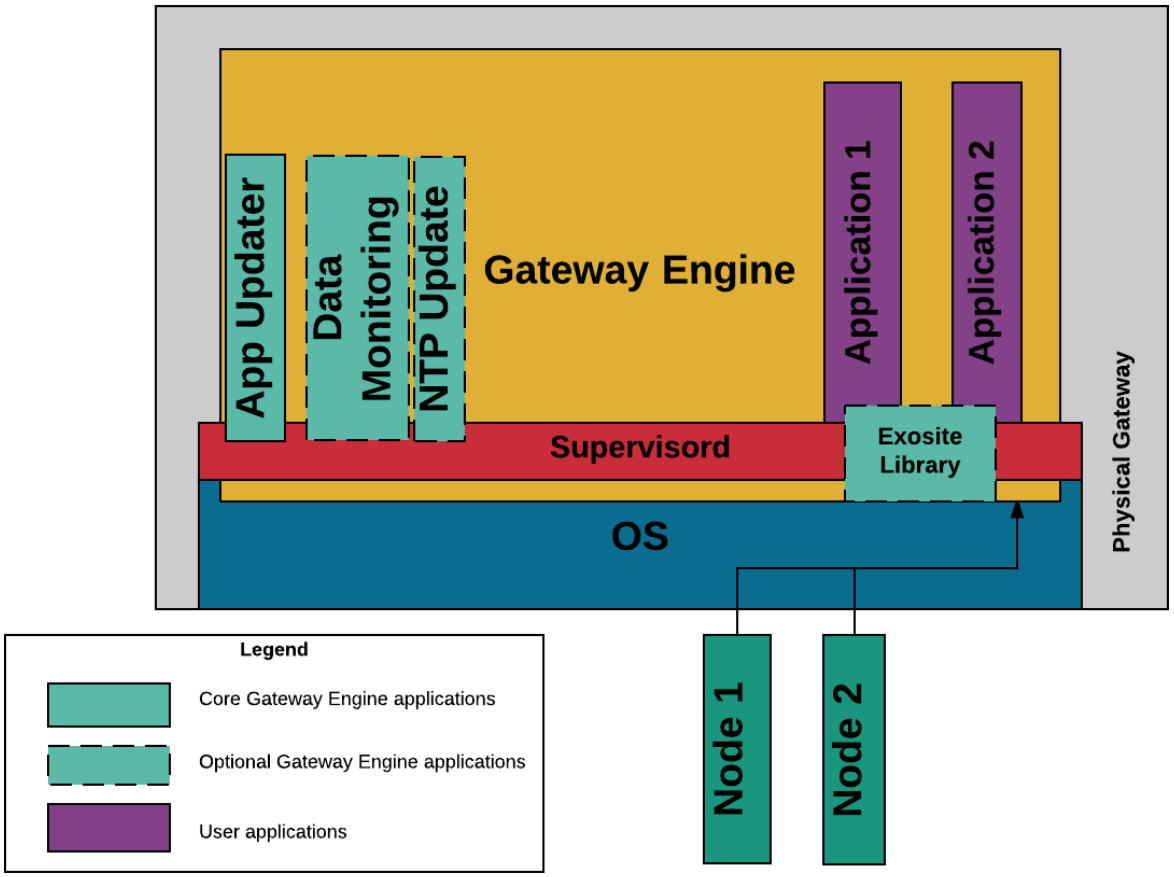
Gateway Engine allows you to remotely update your application by using the Exosite Content Area.

Exosite Interface Module

Gateway Engine comes with a Python module that allows your application to easily communicate with Exosite.

Block Diagram

The following block diagram shows the basic pieces of Gateway Engine:



App Updater Allows remote updates of applications.

Data Monitoring/NTP Update Optional applications that are provided with Exosite. They are used to monitor your data usage and keep the system clock up to date.

Supervisord Makes sure that applications start on boot and restarts them if they die.

Application 1/2 Your business application. You can have 1 or many applications running on in Gateway Engine.

Exosite Library A Python library that is uses to track data usage as well as provide an interface for Python applications to use.

Node 1/2 Your node devices that are communicating with the gateway.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Index