Getting Started - Services

Terminology & Conventions

When using this API documentation, it's good to have a handle on some of the terminology we use in the Services. Here are a few things that are good to know as you're looking at the API endpoints:

Device

This is a class of device defined by a device publisher or manufacturer. Think of it like the model of a car.

Device Instance

This is a specific device that is registered with the platform that is feeding data into the system. Think of it as YOUR car.

Rule

A rule is, well, a rule. You set it up so that if some condition is met, something else happens. In other words, when a rule gets broken, we can alert you. Rules also have the ability to have actions designated to take if the condition in the rule is not met. Either way, you're covered.

Device Reports

This is all of the data that's inbound to the platform from your device instances. Each transmission recorded is called a "report".

Meter Data

This data gives insight as to how much your data is being consumed and called upon.

Use Your Headers

Whenever you make a call into the Services API, a few request headers must be present and set specifically for a request to be processed:

- Authorization → Basic [base64 encoded API key]:X
- Content-type → application/json
- Accept → application/json

As indicated above, you'll need to base64 encode your API key. You can find your API key in the portal under Admin. It looks something like this:

```
xK37cHcKms4Kjyu3BoEo
```

If you make a `POST` or `PUT` request into the API, you would also need to include the *Params* header with valid JSON as its value. A little more on how that works in the next section.

We're going to need to see some ID.

Sometimes you'll see *id* as a part of the endpoint definition. This doesn't mean that you'll actually put "id" in the URL that you're calling, but rather substitute *id* with the database ID of the entity type that is referenced by the endpoint. For example, look at this endpoint:

```
DELETE …/[URL]/v0/device_instance/id
```

To be able to call this successfully, you would need to know the database ID for the specific device instance you want to delete and plug it in where you see *id*.

An *id* always refers to the database record id. It looks like this:

```
5036b57fe817b3ae27000002
```

So then calling that DELETE endpoint we reference earlier would look like this:

```
DELETE …/[URL]/v0/device_instance/5036b57fe817b3ae27000002
```

A couple of logical questions to ask are, "Why is that ID so long and how do I get to it?" Good questions.
First, the reason that the *id* is so long is because it comes directly from the underlying database for the platform, MongoDB. You can read more about MongoDB here: http://mongodb.org. For some specifics on how MongoDB creates those IDs (called *ObjectID*s in MongoDB-speak), you can check this out: http://docs.mongodb.org/manual/core/object-id.

Now for the second part of the question: How do you get to that *id*? Whenever you query the API, any objects that are passed back include a property called `_id`. That's it. That's the *id* that you need to plug into your API calls.

So if you wanted to create a rule for a specific device type, you would do something like this…

Get all of the device types available to you:

```
GET …/[URL]/v0/device
```

This would return something like this:

```
[
	{
		"_id":"509aafa8bc2a68b272000001",
		"active":true,
		"created_at":"2012-11-07T18:59:52Z",
		"deleted_at":null,
		"exe_path":"/foo",
		"image_url":"http://google.com",
		"init_cmd":"test",
		"init_params":"test",
		"long_description":"This is a model 2000",
		"manual_url":"http://google.com",
		"manufacturer":"vendor",
		"model":"vendor 2000",
		"name":"Vendor 2000",
		"payloads":[
			{
				"_id":"509aafa8bc2a68b272000002",
				"created_at":null,
				"data_type":"Number",
				"key_name":"lat",
				"updated_at":null
			},
			{
				"_id":"50be5ff9123f3ae06a000003",
				"created_at":null,
				"data_type":"Number",
				"key_name":"lng",
				"updated_at":null
			},
			{
				"_id":"50be5ff9123f3ae06a000004",
				"created_at":null,
				"data_type":"Number",
				"key_name":"heading",
				"updated_at":null
			},
			{
				"_id":"50be5ff9123f3ae06a000005",
				"created_at":null,
				"data_type":"Number",
				"key_name":"altitude",
				"updated_at":null
			}
		],
		"rule_ids":[
			"50bfbc42123f3acb44000057",
			"50bfc063123f3ac610000002",
			"50bfc8c4123f3ab167000013",
			"50c01a70123f3af274000011"
		],
		"sample_data":"{}",
		"tag_ids":[],
		"unit_price":"1",
		"updated_at":"2012-12-04T20:41:29Z",
		"user_id":"5073950ebc2a68a53b000001"
	}
```

```
]
```

Now you have what you need to know to create a “Near Rule Condition” for all devices of a certain type. You would do a

```
POST …/[URL]/v0/rule
```

And create some JSON that looks like this to go along with that POST:

```
{
    “devices”: ["5099b21f1d41c8f19e00000b"],
    “device_instances”: [],
    “rule”: {
        “description”: "Alert proximity to my place.",
        “active”: true,
        “condition”:{
            “type”: "near_rule_condition",
            “lat_property”: “lat”,
            “lng_property”: “lng”,
            “comp_lat”: 37.42219130,
            “comp_lng”: -122.08458530,
            “comp_radius”: 5,
            “comp_unit”: “:mi”
        },
        “then_actions”: [
            {
                “type”: "sms_rule_action",
                “send_to”: "14795551111",
                “priority”: "1"
            }
        ],
        “else_actions”: []
    }
}
```

The last thing to cover when it comes to *id*s is that they are always associated with the specific object you’re looking for unless preceded by another entity type and an underscore like this:

```
device_instance_id
```

This would indicate that it’s a foreign key (of sorts) to another object referenced by the object you’re looking at. For an example, look at the return from the query to get all device types available to you:

```
[
      {
            "_id":"509aafa8bc2a68b272000001",
            "active":true,
            "created_at":"2012-11-07T18:59:52Z",
            …some stuff omitted…
            "rule_ids":[
                  "50bfbc42123f3acb44000057",
                  "50bfc063123f3ac610000002",
```

```
                "50bfc8c4123f3ab167000013",
                "50c01a70123f3af274000011"
            ],
            "sample_data":"{}",
            "tag_ids":[],
            "unit_price":"1",
            "updated_at":"2012-12-04T20:41:29Z",
            "user_id":"5073950ebc2a68a53b000001"
      }
]
```

That last value, `user_id`, and the array, `rule_ids`, is telling you that this particular device type is associated with a user with the id `5099b21f1d41c8f19e000003` and a few rules. You could find out a little more information about that user by calling

```
GET …/[URL]/v0/user/5099b21f1d41c8f19e000003
```

Then you would see something like this:

```
{
      "_id":"5099b21f1d41c8f19e000003",
      "company_name":"Sample Company",
      "deleted_at":null,
      "email":"sample@email.com",
      "first_name":"API",
      "last_name":"Guy",
      …
      "phone_number":"+1 (123) 456-7890",
      "publisher_id":null,
      "role":"publisher"
}
```

Well good. Now we can get in touch with the publisher of the device if needed. We could also go take a look at what other rules are being applied to this device type – if they're your rules…which brings us to the next subject…

Do you have permission?

As you might expect, not all users of the Services platform has access to all functionality or all data. There are some rules and regulations based on who you are and what role you have within the system.

First of all, any user of the system must have an account within the system in order to access, produce or edit any data. This is the most basic permission – "Authentication" simply asks, "Are you a registered user?"

After "authentication" the platform has a layer of "authorization" that dictates whether you are allowed to take the action you're trying to take based on your role. The platform has 3 roles defined:

1. Administrator

2. Publisher
3. Consumer

Below is a matrix that shows generally what each role is authorized to do in the system:

	Administrator	Publisher	Consumer
User Management	X		
Customer Management		X	
Device Type Data	X	X	
Device Instance Data			X
Meter Data	X	X	X
Device Instance Reports	X		X
Rule Management			X

Lastly, the platform only allows for interaction with data that is directly attributable to the user making the request. For example, if you are a Consumer, you can't see ALL device report data. You can only see report data that is related to device instances for which you are the registering user.

Query Strings & Operators

When using a few of the general GET API endpoints, you have a lot of freedom to specify what you want back. When passing a query string to one of these endpoints, it's a lot like saying, "give me these things where these fields have these values.

For example, let's look at the following GET request:

```
…/device_instances?created_at=2012-12-
21&device_id=5099b21f1d41c8f19e00000b
```

This request would return an array of device instances created on December 21st, 2012 whose device type was specified by the `device_id` passed in.

Well that's great, but it could be better. What if we wanted all of the device reports for a date range? That would be really useful. Or what about finding where some value is greater than a threshold? Good news. You can do all of that. Consider the following example:

```
…/data/monitor?created_at_between=2012-12-01_2012-12-31
```

This would retrieve all device reports for your device instances for the month of December 2012. Note that when using the `_between` operator, the two values are separated by an underscore. This is the only operator that uses this syntax as all

other operators are single-value comparison. All operators are simply appended to the field they are operating on with an underscore as shown.

Speaking of other operators, here is the complete list of operators and their functions:

Operator	English	Logical comparitor
_gt	Greater than	>
_gte	Greater than or equal to	>=
_lt	Less than	<
_lte	Less than or equal to	<=
_between	Between [value 1_value 2]	>= value 1 && <= value 2