Adventures in Groovy – Part 43: Sending Proactive Emails In A Calculation

If you haven't heard, we now have the ability to execute REST API within a calculation script. Not only does that mean we can interact with other environments and do things in calculations we couldn't do before, we also have the ability to interact with any provider that has REST. You could pull current prices for products from Amazon. You could see how many open box items there are at area Best Buy stores. You could pull in currency rates. That doesn't even touch on the things like DM processes, metadata updates, and application refreshes. You can even send emails!

First, what is REST API?

A RESTful API is an application program interface (API) that uses HTTP requests to GET, PUT, POST and DELETE data. A RESTful API – also referred to as a RESTful web service – is based on representational state transfer (REST) technology, an architectural style and approach to communications often used in web services development.

What this means is you can interact with services through web URLs. When you send an email or purchase a product, you are using HTML Post and Get requests.

How does it work?

REST is basically a way to post (do something) and get(receive something) through website URL calls. There is more to it than that, but that is the basic concept. There are all kinds of services, free and paid, that provide services through REST API.

Since I am all about automation and communication, one thing I thought would be great is if a user runs a calculation and it fails, to proactively notify an admin. Prior to this month, you would have to write something outside of the UI for Planning to do this. This would be reactive. Wouldn't it be better to get an email or text immediately so you can act quickly and diagnose the problem? Heck yeah it would.

The REST API Provider

There are a number of providers that give you the ability to send emails via REST API. A few are free for a limited number of emails per day/month. Mailjet is one of them. I went out and signed up for a free account. It allows you to send up to 6k emails a month, with a maximum of 200 per day. That is cool for my purposes of demoing functionality.

There are a few things you need. First is an encrypted key and private key. That you will get when you create our account. Next, the documentation lays out what needs included in a post (do something) to send an email.

To send an email through mailjet, you post to a url of https://api.mailjet.com/v3/send and pass a body that has the typical information to send an email, like from, to, and subject. The body is formatted as a json object. Json is not super complicated, as it is basically the same as a Groovy map, or an xml format. To focus on the API, I will explain json at a later time.

Send An Email

The first thing that needs built is a connection to the REST API provider. You can create a connection to be reused in Planning (epbcs), but for this example, I am just going to keep it simple and use a connection object with the appropriate parameters. To keep my account private, I remove my keys and replaced them with PublicKey and PrivateKey. Just remember these both need replaced with the actual keys. The status should return 200 if the request is successful.

```
Connection connection = connection("https://api.mailjet.com","PuclicKey","PrivateKey")
println  connection.get().asString().status
```

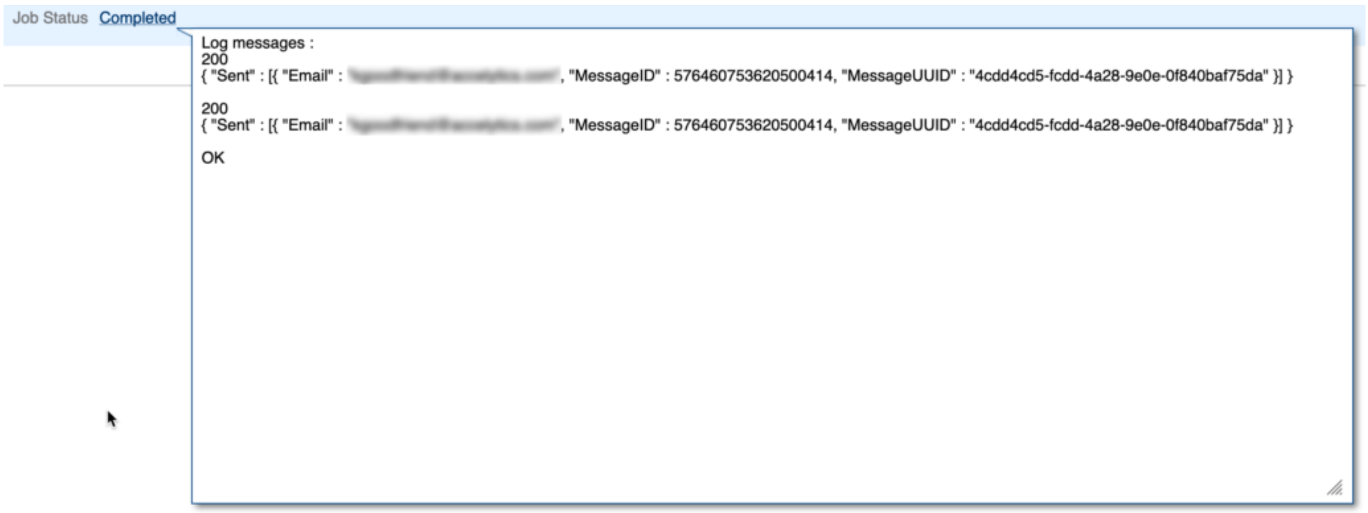
Next, I am going to walk through the body that needs passed. Again, this is basic info that is not surprising. Each element has a property and value. To avoid any more spam in my email account, I replaced the from and to email with invalid emails. These should be changes to include valid from and to emails.

```
def bodyMap = new JSONObject()
bodyMap.put("FromEmail","xxx@in2hyperion.com")
bodyMap.put("FromName","Kyle Goodfriend")
bodyMap.put("Subject","My first email from a Groovy Calculation!")
bodyMap.put("Text-part","This is an email from a calculation")
bodyMap.put("Html-part","<h3>Dear Admin,</H3><br />This is an email from a calculation")
bodyMap.put("To","xxx@in2hyperion.com")
```

Next is the actual request to post (or do something). The path to send emails in the URL is https://api.mailjet.com/v3/send. The only thing we need to add to the post here is the path since the URL is in our connection. The post requires two parameters. First, the request requires one header identifying the content as json. The second parameter is the body that includes the email information, which we have in the bodyMap, which we simply convert to a string.

```
HttpResponse response = connection.post("/v3/send")
.header("content-type","application/json")
.body(bodyMap.toString()).asString()
println response.status
println response.body
println response.statusText
```

This all said, the result is an email to your inbox and a log in the job console that looks like this.



```
Log messages :
200
{ "Sent" : [{ "Email" : [redacted], "MessageID" : 576460753620500414, "MessageUUID" : "4cdd4cd5-fcdd-4a28-9e0e-0f840baf75da" }] }

200
{ "Sent" : [{ "Email" : [redacted], "MessageID" : 576460753620500414, "MessageUUID" : "4cdd4cd5-fcdd-4a28-9e0e-0f840baf75da" }] }

OK
```

Last Call

Well boys and girls, that is all she wrote. That is it. If your calc throws an error, use try/catch/finally and send the appropriate people an email. If the calc is a long running calc, user operation.user to send the person that executed it an email when it finishes. The possibilities are endless but you now have a mechanism to have proactive communication, not reactive and time-consuming effort.

One note about mailjet. I have no affiliation to this service. I have not used it other than to demonstrate this functionality. A FREE account does allow you to send 6k emails a month with 200 at most every day. If you want to use this in a production situation, you likely will need to pay for basic account, which is less than 9 bucks a month.