Using Analysis with NodeJS

6 Tips for Creating and Deploying Scripts using Analysis with NodeJS

Analysis enables you to handle incoming and stored data in realtime. Currently, TagoIO allows to code scripts in node.js when using our servers, but you can use other languages when running your own server (external). Here are six tips to help you to get the most from our Analysis tools using Node.js.

1 Use context.log and analysis console to output errors

When using analysis, your only way to find errors in the process, is through the console. When you run scripts in your machine you may have many ways to debug, but when running the script on Tago, this is the only one.

That said, it's very important that you use context.log to output any errors or important informations from the scripts. This will help you track down any issue you may have.

How to do it:

1. Write in your code:

```
context.log('message here');
```

2 Use the utils lib from SDK to speed up your code

The **Utils** lib from our node.js SDK has different functions to save you time, and to help you write better code.

One of the many basic functions that are presented in many script examples at TagoIO is the *envToObj* function that transforms environment variables into an easily handled object. You will find many other functions to handle data and get device tokens.

How to do it:

1. Import the utils lib in your code by:

```
const tagoUtils = require('tago/utils');
```

2. Use the functions

3 Make use of the Audit log

The Audit log is one of the best developer helper tools that you will find at TagoIO. Through them you can check if an analysis was triggered, when and what data ID triggered it.

You can also check if you had any problem with the email or sms service, that you may never know elsewise.

How to do it:

1. Click on your name at the upper-left corner of the admin.
2. Go to Audit logs option.
3. Select Analysis in Sections to Show.
4. Select your analysis in Analysis dropdown.

4 Avoid timeouts or running out of memory by making your analysis call itself or others

One of the main problems you may face when running big applications or having many users, is your analysis running out of time. This happens when your analysis take more than 2 minutes to stop all it's instructions.

The context variable sent by the analysis contains a parameter called analysis_id, that is your analysis id. Making use of this parameter, together with the Account lib from SDK you can call your analysis again as many times as you want, passing a parameter to it that will arrive through the scope variable.

This is extremely useful when you need to check hundred of users or to manage big data.

How to do it:

1. Import the Account lib:

```
const TagoAccount = require('tago/account');
```

2. Invoke the function to run the analysis:

```
const account = new TagoAccount('token');
account.analysis.run(context.analysis_id, 'parameter'});
```

5 Use the async from JS to make your code more readable

If you are new to node.js or even javascript, you may not understand what *async* is. Many users think they need to use a lot of callbacks or that they are going to have to implement a lot of "*then*" callbacks.

One way to make your code cleaner and more readable is to use *async* functions. It allows you to return the value of a promise directly to a variable, instead of calling the "*then*" function, through the await method.

Example:

```
const TagoDevice = require('tago/device');
const TagoAnalysis = require('tago/analysis');

async function Main(context, scope) {
 const device = new TagoDevice('token');
 const data_from_device = await device.find({ qty: 15 });
 context.log(data_from_device);
}

module.exports = new TagoAnalysis(Main);
```

6 Use environment variables for easy configurations

Inside the analysis configurations in the admin you will find the Environment Variables tab. It allows you to define a key and a value that will always be sent to the analysis when it runs.

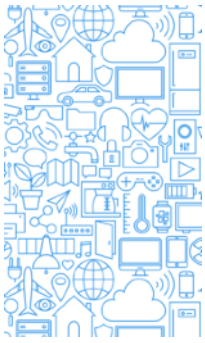
Using the environment variables can help you to create modular scripts that can be easily configured or edited, without the need to directly change the code.

All these environments will be received in the analysis through the context.environment variable.

To easily handle them in the code, you can use the *utils* lib to convert the key/value format to an object format.

Example:

```
function Main(context, scope) {
 const environment =  TagoUtils.env_to_obj(context.environment);
 context.log(environment);
}
```



If you looking for more . . .

- Get access to our complete documentation
- Join our community on Slack
- Ask questions to our technical team or open a ticket
- Watch our tutorials, webinars, and other videos here.