tago-documentation Documentation

Release 2.x.x

Tago LLC

Oct 03, 2017

Contents

1 Device **1**
- 1.1 .info . . . 1
- 1.2 .insert . . . 2
- 1.3 .find . . . 2
- 1.4 .remove . . . 3
- 1.5 .getParams . . . 4
- 1.6 .markParam . . . 5

2 Analysis **7**
- 2.1 Setting Up Analysis . . . 7
- 2.2 context . . . 8
- 2.3 scope . . . 8
- 2.4 Runtime Timeout . . . 8
- 2.5 Running in your machine . . . 8
- 2.6 Tago-Builder and Using Another Packages . . . 8
- 2.7 Services . . . 9

3 Extra **15**
- 3.1 geocoding . . . 15
- 3.2 currency . . . 17
- 3.3 distance . . . 17
- 3.4 weather . . . 18

4 Account **23**
- 4.1 .info . . . 23
- 4.2 .tokenList . . . 24
- 4.3 .tokenCreate . . . 24
- 4.4 .tokenDelete . . . 25
- 4.5 Devices . . . 25
- 4.6 Buckets . . . 30
- 4.7 Actions . . . 34
- 4.8 Analysis . . . 38
- 4.9 Dashboards . . . 42
- 4.10 Widgets . . . 45
- 4.11 notifications . . . 46

CHAPTER 1

Device

In order to modify, add, delete or do anything else with the data inside buckets, it is necessary to use the device function.

To setup an device object, you need a token (that you need to get in our website). Be sure to use tokens with the correct write/read previlegies for the current function that you want to use. For example, a token with only read previlegies can't create, modify or delete anything from a device.

.info

Get all information from the device

Syntax
.info()

Returns
(Promise)

```
const Device = require('tago/device');
const mydev  = new Device('0e479db0-tag0-11e6-8888-790d555b633a');

mydev.info()
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.insert

Insert a new data into a bucket. You can get more information about what information can be passed with insert in our api documentation

Syntax
.insert(/data/)

Arguments
data(object) properties for the new data.

**variable(string): name of the variable. Obrigatory when inserting;*
**value(string): a value for the data (optional);*
**unit(string): a unit for the data, like 'km', or 'F'. The unit may be showed in some widgets (optional);*
**time(string): a time for the data. Default is now;*
**serie(string): a serie for the data. Useful for some widgets when grouping with other data;*
**location(object/geojson): a location object or geojson containing lat and lang;*

Returns
(Promise)

```
const Device = require('tago/device');
const mydev  = new Device('0e479db0-tag0-11e6-8888-790d555b633a');
var data = {
    'variable': 'temperature',
    'unit'    : 'F',
    'value'   : 55,
    'time'    : '2015-11-03 13:44:33',
    'location': {'lat': 42.2974279, 'lng': -85.628292}
};

mydev.insert(data)
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.find

Get a list of data from bucket respecting the query options passed. You can get more information about what information can be passed with .find in our get documentation

Syntax
.find(/filter/)

Arguments

filter(object) filter options when retrieving data. (optional)

**variable(string/array): Filter by variable. If none is passed, get the last data (optional);*

**query(string): Do a specific query. See the query documentation to know what can be passed. (optional)*

**end_date(string): Get data older than a specific date. (optional)*

**start_date(string): Get data newer than a specific date. (optional)*

**qty(number): Number of data to be retrieved. Default is 15. (optional)*

Returns

(Promise)

```
const Device = require('tago/device');
const mydev  = new Device('0e479db0-tag0-11e6-8888-790d555b633a');
var filter = {
    'variable':   'myvar',
    'query':      'last_value',
    'end_date':   '2014-12-25 23:33:22',
    'start_date': '2014-12-20 23:33:22'
};

mydev.find(filter)
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.remove

Remove a data from the bucket. It's possible to remove in three ways: * The last data inserted by the device * The last data inserted by device into a variable * A specific data by it ID

Syntax

.remove(/variable_or_id/, /qty/)

Arguments

variable_or_id(string) a variable name or an specific ID. (optional)

qty(number) specify a number of records to be removed. You can pass "all" to remove all records. Default is 1. (optional)

If no parameter is passed, it will automatically remove the last data inserted by this specific device.

Returns

(Promise)

```
const Device = require('tago/device');
const mydev   = new Device('0e479db0-tag0-11e6-8888-790d555b633a');

mydev.remove()
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

or

```
const Device = require('tago/device');
const mydev   = new Device('0e479db0-tag0-11e6-8888-790d555b633a');

mydev.remove('myvariable')
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

or

```
const Device = require('tago/device');
const mydev   = new Device('0e479db0-tag0-11e6-8888-790d555b633a');

mydev.remove('577d81ac7ee399ef1a6e98da')
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.getParams

Get all params from the device

Syntax
.getParams()

Arguments
filter(boolean) filter options for retrieving the device parameters. (optional)*
**boolean(false)*: *Retrieves all non-sent device parameter;*
**boolean(true)*: *Retrieves all sent device parameter;*

Returns
(Promise)

```
const Device = require('tago/device');
const mydev  = new Device('0e479db0-tag0-11e6-8888-790d555b633a');

mydev.getParams() // you can use getParams(false) or getParams(true)
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.markParam

Marks as read a specific not yet read parameter

Syntax
.markParam(/id/)

Arguments
id(string) using a specific ID. (required)

Returns
(Promise)

```
const Device = require('tago/device');
const mydev  = new Device('0e479db0-tag0-11e6-8888-790d555b633a');

mydev.markParam('59933d82b09301ab13b844ac')
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

CHAPTER 2

Analysis

It's possible to run analysis scripts on your computer, or inside Tago server. In the follow pages, you will be instructed on how to setup an analysis on your computer, use our services, and manage any data from Tago.

If you want to get instructions about how to upload your script or how to use third-party packages inside our server, take a look at admin analysis documentation

Setting Up Analysis

Through analysis, it is possible to insert any calculation and manage your data from Tago in any way you want. We provide some services, such as SMS and email, but you are free to use any third party packages that you need.

To setup an analysis, you first need a analysis token. That can be retrieved from the admin analysis section..

Syntax
new Analysis(/function/, /analysis_token/)

Arguments
function(function) a function to be executed when the analysis runs.
analysis_token(string) analysis token. Only needed if the script will run remotelly (Optional).

```
'use strict';
const Analysis = require('tago/analysis');

//Main function to be executed when the analysis are called
function myanalysis(context, scope) {
    console.log('my context:', context);
    console.log('my scope:', scope);
    //Do anything you want here
```

```
}

module.exports = new Analysis(myanalysis, 'c89f0d50-38e2-11e6-966e-b94d760acc7d');
```

context

As you can setup some predefined parameters in your analysis, it's possible to get these value from the context variable defined in the admin. It is a object, and it comes with follow properties:

PROPERTY	VALUE
environment	All environment variables
token	Token of the analysis
.log(/msg/)	Print a message to the admin console

scope

Every time an action triggers a script, the variable **scope** will be generated. This scope will bring all others variables generated at the same time by the same event. For example, if you submit a form, together with the variable that the script is reading, the scope will return a list of all values/variable input in that form. This allows you to manipulate data in real time, and more easily the new values inserted in your bucket.

Runtime Timeout

Tago Analysis has a mechanism that prevents scripts from being locked in their executions by applying a timeout of 30 seconds. It means that if a script takes more than 30 seconds to be completed, Tago will abort it, and the script will not be completed.

This limitation doesn't apply when running the analyze from your own machine. Check the information below to learn how to run scripts from an external server (e.g. from your own computer).

Running in your machine

You always have the option to run your script from your own machine or from Tago server without any technical difference. When running the script from your machine, you will need to install all the packages used by your analysis by using the command **npm install mypackage**.

Be sure to set your analysis configuration with the option to run the script from "external". And finally, get the analysis token from the same configuration screen, and put it on the second parameter when calling **new Analysis**. Check out this example:

module.exports = new Analysis(myanalysis, 'c89f0d50-38e2-11e6-966e-b94d760acc7d');

Tago-Builder and Using Another Packages

When you are programming, it can be useful to use another packages inside your code; Or you may want to organize your project using *require* and *subfoulders*.

Tago is friendly with some packages:
* **moment** and **moment-timezone**
* **lodash**
* **co**
* **async**
* **axios**
* **crypto**
* **Tago** itself

So you don't need to generate a build if you are using **only** them.

Also, Tago only accepts one single .js file when uploading your script to our servers. ago provides a builder CLI that can build your entire project and generate a single .js file with the whole code. You can access the repository clicking here

To use our Tago-Builder, follow the following steps:

1. **Type** in your terminal **'npm install -g tago-builder'**
2. **Wait** it for the installation to be completed
3. **Type** in your terminal **'tago-builder 'my script'.js 'new name'.tago.js** *(the last parameter is optional)*.
4. **Upload** the generated **'my script'.tago.js** file to **Tago**.

If everything is okay, a new file called 'my script'.tago.js will be generated. Now you can upload this file to Tago!

Services

We provide some functions that can greatly help your application. When creating a analysis, you are can use Tago services on your own, just make sure you understand the policies and cost associate with the usage.

When setting up a service, you need to pass an analysis-token. For convenience, the context returns a property token that you can use to setup a service object.

```
'use strict';
const Analysis = require('tago/analysis');
const Services = require('tago/services');

//Main function to be executed when the analysis are called
function myanalysis(context, scope) {
    //Setting up a SMS service
    const sms = new Services(context.token).sms;

}

module.exports = new Analysis(myanalysis, 'c89f0d50-38e2-11e6-966e-b94d760acc7d');
```

sms

You can configure the system to send SMS directly from your analysis to yourself or your customers. Another option is to use the Actions to send SMS.

Some costs may occur when using the SMS service, which varies based on the country of operation. Check pricing, terms of use, and your plan before using the SMS service.

.send

Whenever you need to send a sms, use .send function.

Syntax
.send(/to/, /message/)

Arguments
to(string) A string with a phone number. If not sending to the USA, you have to add the country code, (+55) for Brazil, for example.
message(string) message to be sent. Use "n" to breakline. (optional)

Returns
(Promise)

```
'use strict';
const Analysis = require('tago/analysis');
const Services = require('tago/services');

//Main function to be executed when analysis are called
function myanalysis(context, scope) {
    const sms = new Services(context.token).sms;

    const to      = '2693856214';
    const message = 'Hi! This is a sms example sent from Tago. \nWith a breakline in␣
↪the sms message.';

    sms.send(to, message).then(console.log).catch(console.log);
    //Print "Sending";

}

module.exports = new Analysis(myanalysis, 'c89f0d50-38e2-11e6-966e-b94d760acc7d');
```

email

Email service allows you to send e-mail through your analysis. Cost may occur when using the e-mail service.

.send

Whenever you need to send an email, use .send function.

Syntax
.send(/to/, /subject/, /message/, /from/, /attachment/)

Arguments
to(string) E-mail address which will receive the email.
subject(string) Subject of the email;
*message(string) message to be sent. Use "
" to breakline.*
from(string) E-mail address for the receiver to reply. Default is tago@tago.io (optional);
attachment(json) Send an attachment with the email (optional);
archive Can be anything: binary, string, number...;
filename(string) Name of the archive with extension. Example: document.txt;

Returns
(Promise)

```
'use strict';
const Analysis = require('tago/analysis');
const Services = require('tago/services');

//Main function to be executed when the analysis are called
function myanalysis(context, scope) {
    const email = new Services(context.token).email;

    const to      = 'myuser@gmail.com';
    const subject = 'E-mail example';
    const message = 'Hi! This is an email example. \nWith a breakline in the email␣
↪message.';
    const from    = 'me@gmail.com';
    const attachment = {
        archive: 'This is a txt archive',
        filename: 'document.txt'
    };

    email.send(to, subject, message, from, attachment).then(console.log).
↪catch(console.log);
    //Print "Sending";

}

module.exports = new Analysis(myanalysis, 'c89f0d50-38e2-11e6-966e-b94d760acc7d');
```

MQTT

This option gives you a lot of flexibility to interpret any kind of data depending on your application. You can send any data format with any content to this topic, your data will go directly to your Analysis inside the scope on the first position of the array. The data will not be stored automatically, your script need to take care of it.

You can read more about MQTT on Tago in our MQTT documentation

.send

Use this topic when you want to send a payload data in any format to be first parsed by a specific script.

Syntax
.publish(/topic/, /message/)

Arguments
topic(string) Topic of the message.
message(string) message to be sent.
bucket(string) bucket id to receive the message. (optional)

Returns
(Promise)

```
'use strict';
const Analysis = require('tago/analysis');
const Services = require('tago/services');

//Main function to be executed when the analysis are called
function myanalysis(context, scope) {
    const MQTT = new Services(context.token).MQTT;

    const topic   = 'my topic';
    const message = 'new message';

    MQTT.publish(topic, message).then(console.log).catch(console.log);
    //Print "Sending";
}

module.exports = new Analysis(myanalysis, 'c89f0d50-38e2-11e6-966e-b94d760acc7d');
```

Notification

Sometimes you may want to send an alert to the account through notification system. You can do it in three ways: pointing to a dashboard, to a bucket or just a notification to the account itself.

When pointing to a dashboard or a bucket, the account owner and anyone he shared the dashboard/bucket will receive the notification.

.send

Use this topic to send a notification.

Syntax
.send(/title/, /message/, /ref_id/)

Arguments
title(string) Title of the message.
message(string) message to be sent.
ref_id(string) dashboard/bucket id that your notification will point to. (optional)

Returns
(Promise)

```
'use strict';
const Analysis = require('tago/analysis');
const Services = require('tago/services');

//Main function to be executed when the analysis are called
function myanalysis(context, scope) {
    const Notification = new Services(context.token).Notification;

    const title   = 'my title';
    const message = 'new message';
    const ref_id = '5915e4a302a0a7002f2a0960'; //bucket id

    Notification.send(title, message, ref_id).then(console.log).catch(console.log);
    //Print "Notification sent";
}

module.exports = new Analysis(myanalysis, 'c89f0d50-38e2-11e6-966e-b94d760acc7d');
```

CHAPTER 3

Extra

Tago support 3party API's to make your life easily, like Google Maps and Weather service. All 3party services below is free to use with a free token that you can get in the owner's website.

geocoding

Whenever you need to get a geolocation (lat/lon) based on a valid address, or vice versa. Use geocoding function. Google Geocoding API docs: https://developers.google.com/maps/documentation/geocoding/

.getGeolocation

Convert the address to a valid geolocation, if it exists.

Syntax
.getGeolocation(/address/)

Arguments
address(string) A valid address.

Returns
(Promise)

```
'use strict';
const Analysis = require('tago/analysis');
const Extra = require('tago/extra');
```

```
//Main function to be executed when an analysis is called
function myanalysis(context, scope) {
    const api_key = 'AIzbSyCLOZEH4go819yAyszUqddIiKZs2-GpJaE'; // API that you can␣
↪get in the google website
    const geocoding = new Extra('geocoding', api_key);

    const address = '1017 Main Campus Dr, Raleigh, NC 27606, USA';

    geocoding.getGeolocation(address).then(console.log).catch(console.log);
    //Print [-78.6772532,35.7704823];
}

module.exports = new Analysis(myanalysis, 'c89f0d50-38e2-11e6-966e-b94d760acc7d');
```

.getAddress

Convert a valid geolocation to an address, if it exists.

Syntax
.getAddress(/geolocation/)

Arguments
geolocation(string) A valid geolocation.

Returns
(Promise)

```
'use strict';
const Analysis = require('tago/analysis');
const Extra = require('tago/extra');

//Main function to be executed when an analysis is called
function myanalysis(context, scope) {
    const api_key = 'AIzbSyCLOZEH4go819yAyszUqddIiKZs2-GpJaE'; // API that you can␣
↪get in the google website
    const geocoding = new Extra('geocoding', api_key);

    const geolocation = '35.7704823,-78.6772532';

    geocoding.getAddress(geolocation).then(console.log).catch(console.log);
    //Print '1017 Main Campus Dr, Raleigh, NC 27606, USA';
}

module.exports = new Analysis(myanalysis, 'c89f0d50-38e2-11e6-966e-b94d760acc7d');
```

currency

Check several currencies in real-time, and the historical exchange rates for more than 168 countries. Currency API: https://currencylayer.com/

.convert

Return the current exchange rate of one currency to another one.

Syntax
.convert(/origins/, /destinations/, /language/, /mode/)

Arguments
from(string) convert from. See 'supported currencies <https://currencylayer.com/currencies>'_ for more information.
to(string) convert to.

Returns
(Promise)

```
'use strict';
const Analysis = require('tago/analysis');
const Extra = require('tago/extra');

//Main function to be executed when the analysis is called
function myanalysis(context, scope) {
    const api_key = '0aa94a3590d5068eb6830d1bf2222d21-GpJaE'; // API that you can get␣
↪in the currencylayer website
    const currency = new Extra('currency', api_key);

    const from = 'USD';
    const to   = 'BRL';

    currency.convert(from, to).then(console.log).catch(console.log);
    //Print Example: 3.29883848
}

module.exports = new Analysis(myanalysis, 'c89f0d50-38e2-11e6-966e-b94d760acc7d');
```

distance

Whenever you need to calculate the distance between two points use distance service. Google Distance API docs: https://developers.google.com/maps/documentation/distance-matrix/intro

.measure

Measure is a service that provides the travel distance and time for a matrix of origins and destinations.

Syntax
.measure(/origins/, /destinations/, /language/, /mode/)

Arguments
origins(array) An array of origins, can be string location or geojson..
destinations(array) An array of destinations, can be string location or geojson.
language(string) Set a language. Default is 'EN'. See 'language support <https://developers.google.com/maps/faq#languagesupport>'_ for more information. (optional)
mode(string) For the calculation of distances, you may specify the transportation mode to use. By default, distances are calculated for driving mode. See the 'travel modes <https://developers.google.com/maps/documentation/distance-matrix/intro#travel_modes>'_ supported for more information.

Returns
(Promise)

```
'use strict';
const Analysis = require('tago/analysis');
const Extra = require('tago/extra');

//Main function to be executed when analysis are called
function myanalysis(context, scope) {
    const api_key = 'AIzbSyCLOZEH4go819yAyszUqddIiKZs2-GpJaE'; // API that you can
↪get in the google website
    const distance = new Extra('distance', api_key);

    const origins      = [ "New York, NY, USA" ];
    const destinations = [ "Washington, DC, USA" ];
    const language     = 'EN';
    const mode         = 'driving';

    distance.measure(origins, destinations, language, mode).then(console.log).
↪catch(console.log);
    //Print
    //TODO; PUT PRINT HERE;
}

module.exports = new Analysis(myanalysis, 'c89f0d50-38e2-11e6-966e-b94d760acc7d');
```

weather

Whenever you need to get weather conditions around the world, use weather service. Wunderground API: https://www.wunderground.com/weather/api/

.current

Get the current weather conditions.

Syntax

.current(/query/, /full/, /language/)

Arguments
query(string) It can be address, zipcode or geojson.
full(boolean) Set to get response with full description. Default is false. (optional)
language(string) Set the language. Default is 'EN'. See 'language support <https://www.wunderground.com/weather/api/d/docs?d=language-support>'_ for more information. (optional)

Returns
(Promise)

```
'use strict';
const Analysis = require('tago/analysis');
const Extra = require('tago/extra');

//Main function to be executed when the analysis is called
function myanalysis(context, scope) {
    const api_key = 'c5e1c5e9dd23967a'; // API that you can get in the wunderground␣
↪website
    const weather = new Extra('weather', api_key);

    const query = '1017 Main Campus Dr, Raleigh, NC 27606, USA'; //address
    //or
    query = '35.7704823,-78.6772532'; //geolocation
    //or
    query = '27605'; //zipcode

    const full     = false;
    const language = "EN"

    weather.current(query, full, language).then(console.log).catch(console.log);
    //Print {"station_id":"KNCRALEI48","observation_time":"Last Updated on July 8,␣
↪5:40 PM EDT","observation_time_rfc822":"Fri, 08 Jul 2016 17:40:04 -0400",
↪"observation_epoch":"1468014004","local_time_rfc822":"Fri, 08 Jul 2016 17:42:43 -
↪0400","local_epoch":"1468014163","local_tz_short":"EDT","local_tz_long":"America/
↪New_York","local_tz_offset":"-0400","weather":"Partly Cloudy","temperature_string":
↪"88.9 F (31.6 C)","temp_f":88.9,"temp_c":31.6,"relative_humidity":"68%","wind_string
↪":"Calm","wind_dir":"North","wind_degrees":-9999,"wind_mph":0,"wind_gust_mph":0,
↪"wind_kph":0,"wind_gust_kph":0,"pressure_mb":"1011","pressure_in":"29.86","pressure_
↪trend":"-","dewpoint_string":"77 F (25 C)","dewpoint_f":77,"dewpoint_c":25,"heat_
↪index_string":"102 F (39 C)","heat_index_f":102,"heat_index_c":39,"windchill_string
↪":"NA","windchill_f":"NA","windchill_c":"NA","feelslike_string":"102 F (39 C)",
↪"feelslike_f":"102","feelslike_c":"39","visibility_mi":"10.0","visibility_km":"16.1
↪","solarradiation":"--","UV":"3","precip_1hr_string":"0.00 in ( 0 mm)","precip_1hr_
↪in":"0.00","precip_1hr_metric":" 0","precip_today_string":"0.00 in (0 mm)","precip_
↪today_in":"0.00","precip_today_metric":"0","icon":"partlycloudy","nowcast":""}";
}

module.exports = new Analysis(myanalysis, 'c89f0d50-38e2-11e6-966e-b94d760acc7d');
```

.forecast

Returns a summary of the weather forecast for the next 10 days. This includes high and low temperatures, a string text forecast and other conditions.

Syntax
.forecast(/query/, /full/, /language/)

Arguments
query(string) It can be address, zipcode or geojson.
full(boolean) Set to get the response with full description. Default is false. (optional)
language(string) Set the language. Default is 'EN'. See 'language support <https://www.wunderground.com/weather/api/d/docs?d=language-support>'_ for more information. (optional)

Returns
(Promise)

```
'use strict';
const Analysis = require('tago/analysis');
const Extra = require('tago/extra');

//Main function to be executed when the analysis is called
function myanalysis(context, scope) {
    const api_key = 'c5e1c5e9dd23967a'; // API that you can get in the wunderground␣
↪website
    const weather = new Extra('weather', api_key);

    const query = '1017 Main Campus Dr, Raleigh, NC 27606, USA'; //address
    //or
    query = '35.7704823,-78.6772532'; //geolocation
    //or
    query = '27605'; //zipcode

    const full     = false;
    const language = "EN"

    weather.forecast(query, full, language).then(console.log).catch(console.log);
    //Print array of 'current weather' for every day in the next 10 days;
}

module.exports = new Analysis(myanalysis, 'c89f0d50-38e2-11e6-966e-b94d760acc7d');
```

.history

Returns a summary of the weather conditions for the last 10 days. This includes high and low temperatures, a string text and other conditions.

Syntax
.history(/date/, /query/, /full/, /language/)

Arguments
date(string) a past date.
query(string) It can be address, zipcode or geojson.
full(boolean) Set to get response with full description. Default is false. (optional)
language(string) Set the language. Default is 'EN'. See 'language support <https://www.wunderground.com/weather/api/d/docs?d=language-support>'_ for more information. (optional)

Returns
(Promise)

```
'use strict';
const Analysis = require('tago/analysis');
const Extra = require('tago/extra');

//Main function to be executed when the analysis is called
function myanalysis(context, scope) {
    const api_key = 'c5e1c5e9dd23967a'; // API that you can get in the wunderground website
    const weather = new Extra('weather', api_key);

    const date  = '2016-07-07';

    const query = '1017 Main Campus Dr, Raleigh, NC 27606, USA'; //address
    //or
    query = '35.7704823,-78.6772532'; //geolocation
    //or
    query = '27605'; //zipcode

    const full     = false;
    const language = "EN"

    weather.history(date, query, full, language).then(console.log).catch(console.log);
    //Print array of 'current weather' for every day until reachs a specified date in the past;
}

module.exports = new Analysis(myanalysis, 'c89f0d50-38e2-11e6-966e-b94d760acc7d');
```

.alert

Returns the short name description, expiration time and a long text description of a severe alert, if one has been issued for the searched location.

Syntax
.alert(/query/, /full/, /language/)

Arguments
query(string) It can be address, zipcode or geojson.
full(boolean) Set to get response with full description. Default is false. (optional)
language(string) Set a language. Default is 'EN'. See 'language support <https://www.wunderground.com/weather/api/d/docs?d=language-support>'_ for more information. (optional)

Returns
(Promise)

```
'use strict';
const Analysis = require('tago/analysis');
const Extra = require('tago/extra');

//Main function to be executed when the analysis is called
function myanalysis(context, scope) {
    const api_key = 'c5e1c5e9dd23967a'; // API that you can get in the wunderground
↪website
    const weather = new Extra('weather', api_key);

    const query = '1017 Main Campus Dr, Raleigh, NC 27606, USA'; //address
    //or
    query = '35.7704823,-78.6772532'; //geolocation
    //or
    query = '27605'; //zipcode

    const full     = false;
    const language = "EN"

    weather.alert(query, full, language).then(console.log).catch(console.log);
    //Print array of the several alerts in the last days;
}

module.exports = new Analysis(myanalysis, 'c89f0d50-38e2-11e6-966e-b94d760acc7d');
```

CHAPTER 4

Account

In order to modify information in the account, dashboard, bucket, device and any other settings, it is necessary to use the device functions.

To setup an account object, you need a token that you need to get in our admin website. Make sure to use tokens with the correct write/read previlegies for the current function that you want to use. For example, a token with only read previlegies can't create, modify or delete anything from an account.

.info

Get all account information

Syntax
.info()

Returns
(Promise)

```
const Account = require('tago/account');
const myacc   = new Account('0e479db0-tag0-11e6-8888-790d555b633a');

myacc.info()
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.tokenList

Get all tokens from the account

Syntax
.tokenList()

Returns
(Promise)

```
const Account = require('tago/account');
const myacc   = new Account('0e479db0-tag0-11e6-8888-790d555b633a');

myacc.tokenList()
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.tokenCreate

Generate and retrieve a new token for the account

Syntax
.tokenCreate()

Arguments
data(object) options for the new token.
**name(string): a name for the token;*
**expire_time(string): Time when token should expire. It will be randomly generated if not included.*

Returns
(Promise)

```
const Account = require('tago/account');
const myacc   = new Account('0e479db0-tag0-11e6-8888-790d555b633a');

myacc.tokenCreate({"name":"My First Token", "expire_time": New Date()})
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
```

```
        //You can treat errors here
    });
```

.tokenDelete

Delete current token of the account

Syntax
.tokenDelete()

Returns
(Promise)

```
const Account = require('tago/account');
const myacc   = new Account('0e479db0-tag0-11e6-8888-790d555b633a');

myacc.tokenDelete()
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

Devices

Across the account function, it is possible to manage all your devices. Make sure that you use an account token with "write" permission when using functions to create, edit and delete. The Device method is completly different from the class Device, since this one can only manage devices, and can't do anything with data related to the device.

.list

Retrieve a list with all devices from account

Syntax
.list()

Returns
(Promise)

```
const Account = require('tago/account');
const accdevices   = new Account('0e479db0-tag0-11e6-8888-790d555b633a').devices;

accdevices.list()
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.create

Generate and retrieve a new device for the account

Syntax
.create(/data/)

Arguments
data(object) options for the new device.
**name(string): a name for the device;*
**description(string): description for the device. (optional)*
**active(bool): Set if the device will be active. Default True. (optional)*
**visible(bool): Set if the device will be visible. Default True. (optional)*
**configuration_params(array): An array of objects with sent(bool), key(string) and value(string). (optional)*
**tags(array): An array of objects with key and value. (optional)*

Returns
(Promise)
**token: token for the generated device;*
**id: id of the new device;*

```
const Account = require('tago/account');
const accdevices   = new Account('0e479db0-tag0-11e6-8888-790d555b633a').devices;
var data = {
    "name":"My first device",
    "description":"Creating my first device",
    "active":true,
    "visible":true,
    "tags": [
        {"key": "client", "value": "John"}
    ]
    "configuration_params": [
        {"sent": false, "key": "check_rate", "value": 600}
        {"sent": false, "key": "measure_time", "value": 0}
    ]
};
```

```
accdevices.create(data)
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.edit

Modify any property of the device.

Syntax
.edit(/id/, /data/)

Arguments
id(string) reference ID of the device.
data(object) options to be modified in the device.
- **name(string): a name for the device; (optional)*
- **description(string): description for the device. (optional)*
- **active(bool): Set if the device will be active. Default True. (optional)*
- **visible(bool): Set if the device will be visible. Default True. (optional)*
- **tags(array): An array of objects with key and value. (optional)*

Returns
(Promise)

```
const Account    = require('tago/account');
const accdevices = new Account('0e479db0-tag0-11e6-8888-790d555b633a').devices;
var data = {
    "name":"New name for my device",
    "description":"In this way I can change the description too",
    "active":false,
    "visible":true,
    "tags": [
        {"key": "client", "value": "Mark"}
    ]
};

accdevices.edit('576dc932415f403531fd2cf6', data)
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.info

Get information about the device

Syntax
.info(/id/)

Arguments
id(string) reference ID of the device.

Returns
(Promise)

```
const Account    = require('tago/account');
const accdevices = new Account('0e479db0-tag0-11e6-8888-790d555b633a').devices;

accdevices.info('576dc932415f403531fd2cf6')
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.delete

Delete device for the account

Syntax
.delete(/id/)

Arguments
id(string) reference ID of the device.

Returns
(Promise)

```
const Account    = require('tago/account');
const accdevices = new Account('0e479db0-tag0-11e6-8888-790d555b633a').devices;

accdevices.delete('576dc932415f403531fd2cf6')
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
```

```
        //You can treat errors here
    });
```

.tokenList

Retrieve a list of all tokens of the device

Syntax
.tokenList(/id/)

Arguments
id(string) reference ID of the device.

Returns
(Promise)

```
const Account    = require('tago/account');
const accdevices = new Account('0e479db0-tag0-11e6-8888-790d555b633a').devices;

accdevices.tokenList('576dc932415f403531fd2cf6')
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.tokenCreate

Generate and retrieve a new token for the device

Syntax
.tokenCreate(/id/, /data/)

Arguments
id(string) reference ID of the device.
data(object) options for the new token.
**name(string): a name for the token;*
**expire_time(string): Time when token should expire. It will be randomly generated if not included. Accept "never" as value.*
**permission(string): Token permission, should be 'write', 'read' or 'full'.*
**serie_number(string): Serial number of the device. (optional)*
**verification_code(string): Verification code to validate middleware requests. (optional)*
**middleware(string): Middleware or type of the device that will be added.. (optional)*

Returns
(Promise)

```
const Account    = require('tago/account');
const accdevices = new Account('0e479db0-tag0-11e6-8888-790d555b633a').devices;

accdevices.tokenCreate({"name":"My First Token", "expire_time": "never", "permission":
↪"full"})
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.tokenDelete

Delete an token of the Device

Syntax
.tokenDelete(/token/)

Arguments
token(string) reference token.

Returns
(Promise)

```
const Account    = require('tago/account');
const accdevices = new Account('0e479db0-tag0-11e6-8888-790d555b633a').devices;

accdevices.tokenDelete('298d17f0-7061-11e6-ab66-b174d8afb89d')
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

Buckets

Across the account function, it is possible to manage all your buckets. Be sure to use an account token with "write" permissions when using functions like create, edit and delete.

.list

Retrieve a list with all buckets from account

Syntax
.list()

Returns
(Promise)

```
const Account = require('tago/account');
const accbuckets   = new Account('0e479db0-tag0-11e6-8888-790d555b633a').buckets;

accbuckets.list()
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.create

Generate and retrieve a new bucket for the account

Syntax
.create(/data/)

Arguments
data(object) options for the new bucket.
**name(string): a name for the bucket;*
**description(string): description for the bucket. (optional)*
**visible(bool): Set if the bucket will be visible or not. Default True. (optional)*
**tags(array): An array of objects with key and value. (optional)*

Returns
(Promise)
**id: id of the new bucket;*

```
const Account = require('tago/account');
const accbuckets   = new Account('0e479db0-tag0-11e6-8888-790d555b633a').buckets;
var data = {
    "name":"My first bucket",
    "description":"Creating my first bucket",
```

```
    "visible":true,
    "tags": [
        {"key": "client", "value": "Francisco"}
    ]
};

accbuckets.create(data)
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.edit

Modify any property of the bucket.

Syntax
.edit(/id/, /data/)

Arguments
id(string) reference ID of the bucket.
data(object) options to be modified in the bucket.
**name(string): a name for the bucket; (optional)*
**description(string): description for the bucket. (optional)*
**visible(bool): Set if the bucket will be visible or not. Default True. (optional)*
**tags(array): An array of objects with key and value. (optional)*

Returns
(Promise)

```
const Account    = require('tago/account');
const accbuckets = new Account('0e479db0-tag0-11e6-8888-790d555b633a').buckets;
var data = {
    "name":"New name for my bucket",
    "description":"This way I can change the description too",
    "visible":true,
    "tags": [
        {"key": "client", "value": "Leonardo"}
    ]
};

accbuckets.edit('576dc932415f403531fd2cf6', data)
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
```

```
        //You can treat errors here
    });
```

.info

Get information about the bucket

Syntax
.info(/id/)

Arguments
id(string) reference ID of the bucket.

Returns
(Promise)

```
const Account    = require('tago/account');
const accbuckets = new Account('0e479db0-tag0-11e6-8888-790d555b633a').buckets;

accbuckets.info('576dc932415f403531fd2cf6')
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.delete

Delete bucket for the account

Syntax
.delete(/id/)

Arguments
id(string) reference ID of the bucket.

Returns
(Promise)

```
const Account    = require('tago/account');
const accbuckets = new Account('0e479db0-tag0-11e6-8888-790d555b633a').buckets;
```

```
accbuckets.delete('576dc932415f403531fd2cf6')
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

Actions

Across the account function, it is possible to manage all your actions. Be sure to use an account token with "write" permissions when using functions like create, edit and delete.

.list

Retrieve a list with all actions from account

Syntax
.list()

Returns
(Promise)

```
const Account = require('tago/account');
const accactions   = new Account('0e479db0-tag0-11e6-8888-790d555b633a').actions;

accactions.list()
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.create

Generate and retrieve a new action for the account

Syntax
.create(/data/)

Arguments
data(object) options for the new action.
**name(string): a name for the action;*

description(string): description for the action. (optional)
active(bool): True if the action is active or not. Default is true(optional)
when_set_bucket(string): ID reference of the bucket(optional)
when_set_origin(string): ID reference of the origin(optional)
when_set_variable(string): name of the variable to trigger when arrive(optional)
*when_set_condition(string): Condition to trigger the action. Can be * (Any), = (Equal), >= (Greater Equal) etc.. (optional)*
*when_set_value(string): Value to be compared by condition. Set to Null if condition is * (Any). (optional)*
when_reset_bucket(string): ID reference of the bucket(optional)
when_reset_origin(string): ID reference of the origin(optional)
when_reset_variable(string): name of the variable to trigger when arrive(optional)
*when_reset_condition(string): Condition to trigger the action. Can be * (Any), = (Equal), >= (Greater Equal) etc.. (optional)*
*when_reset_value(string): Value to be compared by condition. Set to Null if condition is * (Any). (optional)*
type(string): Type of the action. Can be 'script', 'sms', 'email' or 'post', (optional)
tags(array): An array of objects with key and value. (optional)
If type is script
script(string): Reference id of the analysis..(optional)
If type is sms
to(string): Phone number to be sent.(optional)
message(string): Message to be sent in sms.(optional)
If type is email
to(string): E-mail addres to be sent.(optional)
message(string): Message to be sent in e-mail.(optional)
subject(string): Subject of the e-mail.(optional)

Returns
(Promise)
id: id of the new action;

```
const Account = require('tago/account');
const accactions   = new Account('0e479db0-tag0-11e6-8888-790d555b633a').actions;
var data = {
    "name": "a simple action",
    "description": "trigger when the variable test is higher than 2, and reset it␣
↪when is less than 2",
    "when_reset_bucket": "571920982c452fa00c6af660",
    "when_reset_origin": "571920a5cc7d43a00c642ca1",
    "when_reset_variable": "test",
    "when_reset_condition": "<",
    "when_reset_value": "2",
    "when_set_bucket": "571920982c452fa00c6af660",
    "when_set_origin": "571920a5cc7d43a00c642ca1",
    "when_set_variable": "test",
    "when_set_condition": ">",
    "when_set_value": "2",
    "type": "script",
    "script": "577d4c457ee399ef1a6e6cf6",
    "lock": false,
```

```
    "active": true,
    "tags": [
        {"key":"Trigger", "value":"2"}
    ]
};

accactions.create(data)
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.edit

Modify any property of the action.

Syntax
.edit(/id/, /data/)

Arguments
id(string) reference ID of the action.
data(object) properties to be changed. See '.create'_ to more reference..

Returns
(Promise)

```
const Account    = require('tago/account');
const accactions = new Account('0e479db0-tag0-11e6-8888-790d555b633a').actions;
var data = {
    "name":"New name for my action",
    "description":"In this way I can change the description too",
    "visible":true,
    "tags": [
        {"key": "client", "value": "Mark"}
    ]
};

accactions.edit('576dc932415f403531fd2cf6', data)
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.info

Get information about the action

Syntax
.info(/id/)

Arguments
id(string) reference ID of the action.

Returns
(Promise)

```
const Account    = require('tago/account');
const accactions = new Account('0e479db0-tag0-11e6-8888-790d555b633a').actions;

accactions.info('576dc932415f403531fd2cf6')
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.delete

Delete action for the account

Syntax
.delete(/id/)

Arguments
id(string) reference ID of the action.

Returns
(Promise)

```
const Account    = require('tago/account');
const accactions = new Account('0e479db0-tag0-11e6-8888-790d555b633a').actions;

accactions.delete('576dc932415f403531fd2cf6')
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
```

```
        //You can treat errors here
    });
```

Analysis

Across the account function, it is possible to manage all your analysis. Be sure to use an account token with "write" permissions when using functions like create, edit and delete. The analysis method is completly different from the class analysis, since it only manages the analysis information and not the code that it runs.

.list

Retrieve a list with all analysis from account

Syntax
.list()

Returns
(Promise)

```
const Account = require('tago/account');
const accanalysis   = new Account('0e479db0-tag0-11e6-8888-790d555b633a').analysis;

accanalysis.list()
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.create

Generate and retrieve a new analysis for the account

Syntax
.create(/data/)

Arguments
data(object) options for the new analysis.
**name(string): a name for the analysis;*
**description(string): description for the analysis. (optional)*
**interval(string): time interval for analysis to run. Default is Never;*
**active(bool): Set if the analysis will be active. Default True. (optional)*
**variables(array): Environment variables to be passed when the analysis runs. (optional)*

tags(array): An array of objects with key and value. (optional)

Returns
(Promise)
token: token for the generated analysis;
id: id of the new analysis;

```
const Account = require('tago/account');
const accanalysis   = new Account('0e479db0-tag0-11e6-8888-790d555b633a').analysis;
var data = {
    "name":"My first analysis",
    "description":"Creating my first analysis",
    "active":true,
    "interval": '1 minute',
    "variables": [
        {"key": "max_battery", "value": "3100"}
    ],
    "tags": [
        {"key": "client", "value": "Mark"}
    ]
};

accanalysis.create(data)
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.edit

Modify any property of the analysis.

Syntax
.edit(/id/, /data/)

Arguments
id(string) reference ID of the analysis.
data(object) options to be modified in the analysis.
name(string): a name for the analysis; (optional)
description(string): description for the analysis. (optional)
interval(string): time interval for analysis to run. Default is Never;
active(bool): Set if the analysis will be active. Default True. (optional)
variables(array): Environment variables to be passed when the analysis runs. (optional)
tags(array): An array of objects with key and value. (optional)

Returns

(Promise)

```
const Account    = require('tago/account');
const accanalysis = new Account('0e479db0-tag0-11e6-8888-790d555b633a').analysis;
var data = {
    "name":"New name for my analysis",
    "description":"In this way I can change the description too",
    "active":false,
    "interval": '2 minutes',
    "variables": [
        {"key": "max_battery", "value": "3000"}
    ],
    "tags": [
        {"key": "client", "value": "Mark"}
    ]
};

accanalysis.edit('576dc932415f403531fd2cf6', data)
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.info

Get information about the analysis

Syntax
.info(/id/)

Arguments
id(string) reference ID of the analysis.

Returns
(Promise)

```
const Account    = require('tago/account');
const accanalysis = new Account('0e479db0-tag0-11e6-8888-790d555b633a').analysis;

accanalysis.info('576dc932415f403531fd2cf6')
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.delete

Delete analysis for the account

Syntax
.delete(/id/)

Arguments
id(string) reference ID of the analysis.

Returns
(Promise)

```
const Account    = require('tago/account');
const accanalysis = new Account('0e479db0-tag0-11e6-8888-790d555b633a').analysis;

accanalysis.delete('576dc932415f403531fd2cf6')
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.run

Force Analysis to run immediately

Syntax
.run(/id/)

Arguments
id(string) reference ID of the analysis.

Returns
(Promise)

```
const Account    = require('tago/account');
const accanalysis = new Account('0e479db0-tag0-11e6-8888-790d555b633a').analysis;

accanalysis.run('576dc932415f403531fd2cf6')
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
```

```
        //You can treat errors here
    });
```

Dashboards

Across the account function, it is possible to manage all your dashboards. Be sure to use an account token with "write" permissions when using functions like create, edit and delete.

.list

Retrieve a list with all dashboards from account

Syntax
.list()

Returns
(Promise)

```
const Account = require('tago/account');
const accdashboards   = new Account('0e479db0-tag0-11e6-8888-790d555b633a').
↪dashboards;

accdashboards.list()
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.create

Generate and retrieve a new dashboard for the account

Syntax
.create(/data/)

Arguments
data(object) options for the new dashboard.
**label(string): a label for the dashboards;*
**arrangement(array): array of objects with arrangement of the widget inside the dashboard. (optional)*
**widget_id(string): id of the widget*
**x(number): position x of the widget. 1 to 4;*
**y(number): position y of the widget. 1 to 20*

*width(number): width of the widget. 1 to 4
*height(number): height of the widget. 1 to 20
*tags(array): An array of objects with key and value. (optional)

Returns
(Promise)
*token: token for the generated dashboard;
*id: id of the new dashboard;

```
const Account = require('tago/account');
const accdashboards   = new Account('0e479db0-tag0-11e6-8888-790d555b633a').
↪dashboards;
var data = {
    "label":"My first dashboard",
    "arrangement": [
        {"widget_id": "577c28d269d2861f1b2e93b8", "x":0, "y":0, "width":2, "height":3␣
↪}
    ],
    "tags": [
        {"key": "client", "value": "Mark"}
    ]
};

accdashboards.create(data)
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.edit

Modify any property of the dashboards.

Syntax
.edit(/id/, /data/)

Arguments
id(string) reference ID of the dashboards.
data(object) options to be modified in the dashboards.
*label(string): a label for the dashboards;
*arrangement(array): array of objects with arrangement of the widgest inside the dashboard. (optional)
*widget_id(string): id of the widget
*x(number): position x of the widget. 1 to 4;
*y(number): position y of the widget. 1 to 20
*width(number): width of the widget. 1 to 4
*height(number): height of the widget. 1 to 20

tags(array): An array of objects with key and value. (optional)

Returns
(Promise)

```
const Account    = require('tago/account');
const accdashboards = new Account('0e479db0-tag0-11e6-8888-790d555b633a').dashboards;
var data = {
    "label":"New name for my dashboards",
};

accdashboards.edit('877c28d269d2861f1b2e96b8', data)
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.info

Get information about the dashboards

Syntax
.info(/id/)

Arguments
id(string) reference ID of the dashboards.

Returns
(Promise)

```
const Account    = require('tago/account');
const accdashboards = new Account('0e479db0-tag0-11e6-8888-790d555b633a').dashboards;

accdashboards.info('877c28d269d2861f1b2e96b8')
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.delete

Delete dashboards for the account

Syntax
.delete(/id/)

Arguments
id(string) reference ID of the dashboards.

Returns
(Promise)

```
const Account    = require('tago/account');
const accdashboards = new Account('0e479db0-tag0-11e6-8888-790d555b633a').dashboards;

accdashboards.delete('877c28d269d2861f1b2e96b8')
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

Widgets

Inside dashboards, you need widgets to show and control information inside buckets. Every widget have their data slighty different from each other, to know how do they work

.create

Generate and retrieve a new dashboard for the account

Syntax
.create(/data/)

Arguments
data(object) options for the new dashboard.
**label(string): a label for the dashboards;*
**arrangement(array): array of objects with arrangement of the widget inside the dashboard. (optional)*
**widget_id(string): id of the widget*
**x(number): position x of the widget. 1 to 4;*
**y(number): position y of the widget. 1 to 20*
**width(number): width of the widget. 1 to 4*
**height(number): height of the widget. 1 to 20*
**tags(array): An array of objects with key and value. (optional)*

Returns
(Promise)

token: token for the generated dashboard;
id: id of the new dashboard;

```
const Account = require('tago/account');
const accdashboards   = new Account('0e479db0-tag0-11e6-8888-790d555b633a').
↪dashboards;
var data = {
    "label":"My first dashboard",
    "arrangement": [
        {"widget_id": "577c28d269d2861f1b2e93b8", "x":0, "y":0, "width":2, "height":3␣
↪}
    ],
    "tags": [
        {"key": "client", "value": "Mark"}
    ]
};

accdashboards.create(data)
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

notifications

All accounts have an notification system, where you can see important alerts and accept/refuse share of dashboards, profiles and buckets.

.list

Retrieve a list with all notifications from account

Syntax
.list()

Returns
(Promise)
result(array): Array list of notifications;

```
const Account = require('tago/account');
const notifications   = new Account('0e479db0-tag0-11e6-8888-790d555b633a').
↪notifications;

notifications.list()
```

```
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.markAsRead

Mark a notification as read/ignored.

Syntax
.markAsRead(/id_list/)

Arguments
**id_list(array)*: *array of notification ids;*

Returns
(Promise)
result: *Notifications marked as read;*

```
const Account = require('tago/account');
const notifications   = new Account('0e479db0-tag0-11e6-8888-790d555b633a').
↪notifications;

const id_list = ['5915e4a302a0a7002f2a0960', '4915e4a302a0a7002f3a0982']
notifications.markAsRead(id_list)
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.accept

Accept the notification if it has a condition.

Syntax
.accept(/notification_id/)

Arguments
**notification_id(string)*: *ID of the notification;*

Returns

(Promise)
**result*: *Notification succesfully accepted;*

```
const Account = require('tago/account');
const notifications   = new Account('0e479db0-tag0-11e6-8888-790d555b633a').
↪notifications;

const notification_id = '5915e4a302a0a7002f2a0960'
notifications.accept(notification_id)
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```

.refuse

Refuse the notification if it has a condition.

Syntax
.refuse(/notification_id/)

Arguments
**notification_id(array)*: *ID of the notification;*

Returns
(Promise)
**result*: *Notification succesfully refused;*

```
const Account = require('tago/account');
const notifications   = new Account('0e479db0-tag0-11e6-8888-790d555b633a').
↪notifications;

const notification_id = '5915e4a302a0a7002f2a0960'
notifications.refuse(notification_id)
    .then((result) => {
        //You can treat the result here
    })
    .catch((error) => {
        //You can treat errors here
    });
```