tago-documentation Documentation

Release 3.x.x

TagoIO LLC

Jan 10, 2020

Contents

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.

1.1 .info

Get all information from the device

Syntax
.info()

Returns
(Promise)

```
import tago

my_device = tago.Device('DEVICE_TOKEN_HERE')

device_information = my_device.info()
print(device_information)
```

1.2 .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)

```
import tago

my_device = tago.Device('DEVICE_TOKEN_HERE')

data = {
        'variable': 'temperature',
        'unit'    : 'F',
        'value'   : 55,
        'time'    : '2015-11-03 13:44:33',
        'location': {'lat': 42.2974279, 'lng': -85.628292}
}

result = my_device.insert(data)
print(result)
```

1.3 .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)

```
import tago

my_device = tago.Device('DEVICE_TOKEN_HERE')

filter = {
        'variable': 'myvar',
        'query': 'last_value',
        'end_date': '2014-12-25 23:33:22',
        'start_date': '2014-12-20 23:33:22'
}

result = my_device.find(filter)
print(result)
```

1.4 .remove

Remove a data from the bucket using a JSON filter.

Syntax
.remove(/filter/)

Arguments
filter(object) filter options when deleting 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 deleted. Default is 15. (optional)

Returns
(Promise)

```
import tago

my_device = tago.Device('DEVICE_TOKEN_HERE')
```

(continues on next page)

(continued from previous page)

```
filter = {
    'variable': 'myvar',
    'query': 'last_value',
    'end_date':   '2014-12-25 23:33:22',
    'start_date': '2014-12-20 23:33:22'
}
result = my_device.remove(filter)
print(result)
```

or

```
import tago

my_device = tago.Device('DEVICE_TOKEN_HERE')

result = my_device.remove('myvariable')
print(result)
```

or

```
import tago

my_device = tago.Device('DEVICE_TOKEN_HERE')

result = my_device.remove('VARIABLE_ID_HERE')
print(result)
```

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

2.1 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).

```
from tago import Analysis

ANALYSYS_TOKEN = 'ANALYSYS_TOKEN_HERE'

def my_analysis(context, scope):
    context.log('my context', context)
    context.log('my scope', scope)
```

(continues on next page)

(continued from previous page)

```
    # Do anything you want here

Analysis(ANALYSYS_TOKEN).init(my_analysis)
```

2.2 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

2.3 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.

2.4 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).

2.5 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');

2.6 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!

2.7 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.

```
from tago import Analysis
from tago import Services

ANALYSYS_TOKEN = 'ANALYSYS_TOKEN_HERE'

def my_analysis(context, scope):
    sms = Services(ANALYSYS_TOKEN).sms
    # Do anything you want here

Analysis(ANALYSYS_TOKEN).init(my_analysis)
```

2.7.1 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)

```
from tago import Analysis
from tago import Services

ANALYSYS_TOKEN = 'ANALYSYS_TOKEN_HERE'

def my_analysis(context, scope):
    sms = Services(ANALYSYS_TOKEN).sms

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

    # Print response
    print(sms.send(to, message))

    # Do anything you want here

Analysis(ANALYSYS_TOKEN).init(my_analysis)
```

2.7.2 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)

```
from tago import Analysis
from tago import Services

ANALYSYS_TOKEN = 'ANALYSYS_TOKEN_HERE'

def my_analysis(context, scope):
    email = Services(ANALYSYS_TOKEN).email

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

    # Printing response
    print(email.send(to, subject, message, email_origin, attachment, None, None))

    # Do anything you want here

Analysis(ANALYSYS_TOKEN).init(my_analysis)
```

2.7.3 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)

```
from tago import Analysis
from tago import Services

ANALYSYS_TOKEN = 'ANALYSYS_TOKEN_HERE'

def my_analysis(context, scope):
    MQTT = Services(ANALYSYS_TOKEN).MQTT

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

    # Printing response
    print(MQTT.publish(topic, message, None))

    # Do anything you want here

Analysis(ANALYSYS_TOKEN).init(my_analysis)
```

2.7.4 Notification to myself

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)

```
from tago import Analysis
from tago import Services

ANALYSYS_TOKEN = 'ANALYSYS_TOKEN_HERE'

def my_analysis(context, scope):
    notification = Services(ANALYSYS_TOKEN).notification

    title   = 'my title';
    message = 'new message';
    ref_id = 'ID_HERE'; # dashboard/bucket id

    # Printing response
    print(notification.send(title, message, ref_id))

    # Do anything you want here

Analysis(ANALYSYS_TOKEN).init(my_analysis)
```

CHAPTER 3

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.

3.1 .info

Get all account information

Syntax
.info()

Returns
(Promise)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')
account_info = my_account.info()

print(account_info)
```

3.2 .tokenList

Get all tokens from the account

Syntax
.tokenList(profile_id)

Arguments
profile_id(string) id of the profile.

Returns
(Promise)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

result = my_account.profiles.tokenList('PROFILE ID HERE')
print(result)
```

3.3 .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;*
**profile_id(string): profile_id from the profile where token will be created;*
**email(string): account email;*
**password(string): account password;*
**expire_time(string): Relative time when token should expire. It will be randomly generated if not included.*

Returns
(Promise)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')
```

(continues on next page)

(continued from previous page)

```
new_token = { 'name': 'My First Token',
              'profile_id': 'PROFILE ID HERE',
              'email': 'ACCOUNT EMAIL HERE',
              'password': 'ACCOUNT PASSWORD HERE',
              'expire_time': '1 day' }

result = my_account.tokenCreate(new_token)
print(result)
```

3.4 .tokenDelete

Delete current token of the account

.tokenDelete(profile_id, token)

Arguments
profile_id(string) id of the profile.
token(string) profile token.

Returns
(Promise)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

result = my_account.profiles.tokenDelete('PROFILE ID HERE', 'PROFILE TOKEN HERE')
print(result)
```

3.5 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.

3.5.1 .list

Retrieve a list with all devices from account

Syntax
.list()

Returns
(Promise)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')
my_devices = my_account.devices.list()

print(my_devices)
```

3.5.2 .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;*

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

new_device = {
    '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},
```

(continues on next page)

(continued from previous page)

```
        {'sent': False, 'key': 'measure_time', 'value': 0}
    ]
}

result = my_account.devices.create(new_device)
print(result)
```

3.5.3 .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)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

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'}
    ]
}

result = my_account.devices.edit('DEVICE_ID', data)
print(result)
```

3.5.4 .info

Get information about the device

Syntax
.info(/id/)

Arguments
id(string) reference ID of the device.

Returns
(Promise)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

device_info = my_account.devices.info('DEVICE_ID')
print(device_info)
```

3.5.5 .delete

Delete device for the account

Syntax
.delete(/id/)

Arguments
id(string) reference ID of the device.

Returns
(Promise)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

device_info = my_account.devices.delete('DEVICE_ID')
print(device_info)
```

3.5.6 .tokenList

Retrieve a list of all tokens of the device

Syntax
.tokenList(/id/)

Arguments
id(string) reference ID of the device.

Returns
(Promise)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

device_token_list = my_account.devices.tokenList('DEVICE_ID')
print(device_token_list)
```

3.5.7 .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)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

new_token_data = {
    'name': 'My First Token',
    'expire_time': 'never',
    'permission': 'full'
}
```

(continues on next page)

(continued from previous page)

```
result = my_account.devices.tokenCreate('DEVICE_ID', new_token_data)
print(result)
```

3.5.8 .tokenDelete

Delete an token of the Device

Syntax
.tokenDelete(/token/)

Arguments
token(string) reference token.

Returns
(Promise)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

result = my_account.devices.tokenDelete('TOKEN')
print(result)
```

3.6 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.

3.6.1 .list

Retrieve a list with all buckets from account

Syntax
.list()

Returns
(Promise)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

my_buckets = my_account.buckets.list()
print(my_buckets)
```

3.6.2 .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)
**object.bucket: id of the new bucket;*

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

new_bucket_data = {
    'name':'My first bucket',
    'description':'Creating my first bucket',
    'visible': True,
    'tags': [
        {'key': 'client', 'value': 'Francisco'}
    ]
}

result = my_account.buckets.create(new_bucket_data)
print(result)
```

3.6.3 .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)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

data = {
    'name':'New name for my bucket',
    'description':'This way I can change the description too',
    'visible': True,
    'tags': [
        {'key': 'client', 'value': 'Leonard'}
    ]
}

result = my_account.buckets.edit('BUCKET_ID_HERE', data)
print(result)
```

3.6.4 .info

Get information about the bucket

Syntax
.info(/id/)

Arguments
id(string) reference ID of the bucket.

Returns
(Promise)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')
```

(continues on next page)

(continued from previous page)

```
result = my_account.buckets.info('BUCKET_ID_HERE')
print(result)
```

3.6.5 .delete

Delete bucket for the account

Syntax
.delete(/id/)

Arguments
id(string) reference ID of the bucket.

Returns
(Promise)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

result = my_account.buckets.delete('BUCKET_ID_HERE')
print(result)
```

3.7 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.

3.7.1 .list

Retrieve a list with all actions from account

Syntax
.list()

Returns
(Promise)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

my_actions = my_account.actions.list()
print(my_actions)
```

3.7.2 .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;*

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

new_action_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'}
    ]
}

result = my_account.actions.create(new_action_data)
print(result)
```

3.7.3 .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)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

data = {
```

(continues on next page)

(continued from previous page)

```
    'name':'New name for my action',
    'description':'In this way I can change the description too',
    'visible': True,
    'tags': [
        {'key': 'client', 'value': 'Mark'}
    ]
}

result = my_account.actions.edit('ACTION_ID_HERE', data)
print(result)
```

3.7.4 .info

Get information about the action

Syntax
.info(/id/)

Arguments
id(string) reference ID of the action.

Returns
(Promise)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

action_info = my_account.actions.info('ACTION_ID_HERE')
print(action_info)
```

3.7.5 .delete

Delete action for the account

Syntax
.delete(/id/)

Arguments
id(string) reference ID of the action.

Returns
(Promise)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

result = my_account.actions.delete('ACTION_ID_HERE')
print(result)
```

3.8 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.

3.8.1 .list

Retrieve a list with all analysis from account

Syntax
.list()

Returns
(Promise)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

my_analysis = my_account.analysis.list()
print(my_analysis)
```

3.8.2 .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;

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

new_analysis_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'}
    ]
}

result = my_account.analysis.create(new_analysis_data)
print(result)
```

3.8.3 .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)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

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'}
    ]
}

result = my_account.analysis.create(data)
print(result)
```

3.8.4 .info

Get information about the analysis

Syntax
.info(/id/)

Arguments
id(string) reference ID of the analysis.

Returns
(Promise)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

analysis_info = my_account.analysis.info('ANALYSIS_ID_HERE')
print(analysis_info)
```

3.8.5 .delete

Delete analysis for the account

Syntax
.delete(/id/)

Arguments
id(string) reference ID of the analysis.

Returns
(Promise)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

result = my_account.analysis.delete('ANALYSIS_ID_HERE')
print(result)
```

3.8.6 .run

Force Analysis to run immediately

Syntax
.run(/id/)

Arguments
id(string) reference ID of the analysis.

Returns
(Promise)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

# Your scope can be an array with several objects
scope = [
    {
        'variable': 'alarm',
        'value:' 'On'
    }
]

result = my_account.analysis.run('ANALYSIS_ID_HERE', scope)
print(result)
```

3.9 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.

3.9.1 .list

Retrieve a list with all dashboards from account

Syntax
.list()

Returns
(Promise)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

my_dashboards = my_account.dashboards.list()
print(my_dashboards)
```

3.9.2 .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;*

```
import tago
```

(continues on next page)

(continued from previous page)

```
my_account = tago.Account('MY_ACCOUNT_TOKEN')

new_dashboard_data = {
    'label':'My first dashboard',
    'arrangement': [
        {'widget_id': 'WIDGET_ID_HERE', 'x': 0, 'y': 0, 'width': 2, 'height': 3}
    ],
    'tags': [
        {'key': 'client', 'value': 'Mark'}
    ]
}

result = my_account.dashboards.create(new_dashboard_data)
print(result)
```

3.9.3 .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)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

data = {
    'label':'New name for my dashboard',
}

result = my_account.dashboards.edit('DASHBOARD_ID_HERE', data)
print(result)
```

3.9.4 .info

Get information about the dashboards

Syntax
.info(/id/)

Arguments
id(string) reference ID of the dashboards.

Returns
(Promise)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

dashboard_info = my_account.dashboards.info('DASHBOARD_ID_HERE')
print(dashboard_info)
```

3.9.5 .delete

Delete dashboards for the account

Syntax
.delete(/id/)

Arguments
id(string) reference ID of the dashboards.

Returns
(Promise)

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

result = my_account.dashboards.delete('DASHBOARD_ID_HERE')
print(result)
```

3.10 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

3.10.1 .create

Generate and retrieve a new widget for the dashboard

Syntax
.create(/dashboard_id/, /data/)

Arguments
dashboard_id(string) dashboard id for the dashboard.
data(object) options for the new widget.
**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*

Returns
(Promise)

```
import tago

my_account = tago.Account('54d83222-6837-4e9e-8f2e-67de8fce5a8b')

new_widget = {
    'label': 'My first widget',
    'type': 'step_button',
    'data': [{
        'bucket': '5d02be12cac6da0026398ed2',
        'origin': '5d02be12cac6da0026398ed1',
        'timezone': 'America/Sao_Paulo',
        'query': 'last_value',
        'variables': ['humidity']
    }]
}

result = my_account.dashboards.widgets.create('DASHBOARD_ID_HERE', new_widget)
print(result)
```

3.10.2 .edit

Modify any property of the widget.

Syntax
.edit(/dashboard_id/, /widge_id/, /data/)

Arguments
dashboard_id(string) dashboard id for the dashboard.
widge_id(string) widget id for the dashboard.
data(object) options for the new widget.
**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(optional)*
**x(number): position x of the widget. 1 to 4; (optional)*
**y(number): position y of the widget. 1 to 20(optional)*
**width(number): width of the widget. 1 to 4(optional)*
**height(number): height of the widget. 1 to 20(optional)*

Returns
(Promise)

```
import tago

account_dashboards = tago.Account('MY_ACCOUNT_TOKEN').dashboards

data = {
    'label':'New name for my widget',
}

result = account_dashboards.widgets.edit('DASHBOARD_ID_HERE', 'WIDGET_ID_HERE', data)
print(result)
```

3.10.3 .info

Get information about the widget

Syntax
.info(/dashboard_id/, /widge_id/)

Arguments
id(string) reference ID of the dashboard.
id(string) reference ID of the widget.

Returns

(Promise)

```
import tago

account_dashboards = tago.Account('MY_ACCOUNT_TOKEN').dashboards

widget_info = account_dashboards.widgets.info('DASHBOARD_ID_HERE', 'WIDGET_ID_HERE')
print(widget_info)
```

3.10.4 .delete

Delete access widget for the dashboard

Syntax
.delete(/dashboard_id/, /widge_id/)

Arguments
id(string) reference ID of the dashboard.
id(string) reference ID of the widget.
Returns
(Promise)

```
import tago

account_dashboards = tago.Account('MY_ACCOUNT_TOKEN').dashboards

result = account_dashboards.widgets.delete('DASHBOARD_ID_HERE', 'WIDGET_ID_HERE')
print(result)
```

3.11 notifications to myself

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

3.11.1 .list

Retrieve a list with all notifications from account

Syntax
.list()

Returns

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

```
import tago

my_account = tago.Account('MY_ACCOUNT_TOKEN')

# Empty object as params retrive all notifications from account
params = {}

my_notifications = my_account.notifications.list(params)
print(my_notifications)
```

3.11.2 .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;*

```
import tago

my_account = tago.Account('54d83222-6837-4e9e-8f2e-67de8fce5a8b')

# Array of notifications ids, this array can have several notifications ids
notifications_ids_list = ['NOTIFICATION_ID_HERE', 'ANOTHER_NOTIFICATION_ID_HERE']

result = my_account.notifications.markAsRead(notifications_ids_list)
print(result)
```

3.11.3 .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;*

```
import tago

my_account = tago.Account('54d83222-6837-4e9e-8f2e-67de8fce5a8b')

result = my_account.notifications.accept('NOTIFICATION_ID_HERE')
print(result)
```

3.11.4 .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;*

```
import tago

my_account = tago.Account('54d83222-6837-4e9e-8f2e-67de8fce5a8b')

result = my_account.notifications.accept('NOTIFICATION_ID_HERE')
print(result)
```

3.12 TagoRun Users

You can manage your TagoRun and Run Users. In order to modify, add, delete or do anything else with the data inside Run. See more about Tago Run here.

To setup an device object, you need a account-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 Run.

3.12.1 .info

Get all information from the run

Syntax
.info()

Returns
(Promise)

```
import tago

my_account = tago.Account('54d83222-6837-4e9e-8f2e-67de8fce5a8b')

tago_run_info = my_account.run.info()
print(tago_run_info)
```

3.12.2 .listUsers

Retrieve a list with all users from Run

Syntax
.listUsers()

Returns
(Promise)

```
import tago

my_account = tago.Account('54d83222-6837-4e9e-8f2e-67de8fce5a8b')

tago_run_users = my_account.run.listUsers()
print(tago_run_users)
```

3.12.3 .getUserInfo

Get run user information

Syntax
.getUserInfo()

Arguments
**user_id(string)*: *ID of the run user;*

Returns
(Promise)

```
import tago

my_account = tago.Account('54d83222-6837-4e9e-8f2e-67de8fce5a8b')

run_user_info = my_account.run.getUserInfo('RUN_USER_ID_HERE')
print(run_user_info)
```

3.12.4 .userEdit

Modify any property of the Run User.

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

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

Returns
(Promise)

```
import tago

my_account = tago.Account('54d83222-6837-4e9e-8f2e-67de8fce5a8b')

data = {
    'name':'New name for my Run User',
    'tags': [
        {'key': 'client', 'value': 'Mark'}
    ]
```

(continues on next page)

(continued from previous page)

```
}

result = my_account.run.userEdit('RUN_USER_ID_HERE', data)
print(result)
```

3.12.5 .createUser

Create a new Run User.

Syntax
.createUser(/data/)

Arguments
data(object) options to be modified in the run user.
- **name(string): a name for the run user.*
- **email(string): email for the run user.*
- **password(string): password for the run user.*
- **timezone(string): timezone for the run user.*
- **phone(string): phone for the run user. (optional)*
- **company(string): company for the run user. (optional)*
- **active(bool): Set if the run user will be active. Default True. (optional)*
- **tags(array): An array of objects with key and value. (optional)*

Returns
(Promise)

```
import tago

my_account = tago.Account('54d83222-6837-4e9e-8f2e-67de8fce5a8b')

new_user = {
    'name':'John Doe',
    'email': 'jhon@doe.com',
    'password': '123abc',
    'tags': [
        {'key': 'employee', 'value': 'Manager'}
    ],
    'timezone': 'America/Sao_Paulo'
}

result = my_account.run.createUser(new_user)
print(result)
```

3.12.6 .deleteUser

Delete run user

Syntax
.deleteUser()

Arguments
**user_id(string)*: *ID of the run user;*

Returns
(Promise)

```
import tago

my_account = tago.Account('54d83222-6837-4e9e-8f2e-67de8fce5a8b')

result = my_account.run.deleteUser('5d7adfb1f03154001bbd9d78')
print(result)
```

3.13 Notification to users

You can push notification messages directly to the users registered in your Run. See more about notification for users here.

3.13.1 .notificationList

Retrieve a list with all notifications for the Run user

Syntax
.notificationList()

Arguments
**user_id(string)*: *ID of the run user;*

Returns
(Promise)

```
import tago

my_account = tago.Account('54d83222-6837-4e9e-8f2e-67de8fce5a8b')

run_user_notifications_list = my_account.run.notificationList('RUN_USER_ID_HERE')
print(run_user_notifications_list)
```

3.13.2 .notificationEdit

Modify any property of the user push notification.

Syntax
.notificationEdit(/notification_id/, /data/)

Arguments
notification_id(string) reference ID of the notification.
data(object) options to be modified in the notification.
**title(string): a title for the notification. (optional)*
**message(string): message for the notification. (optional)*
**buttons(array of object): phone for the run user. (optional)*
**label(string): label for notification button. (optional)*
**analysis(string): analysis_id for notification button. This analysis is run when the button is pressed. (optional)*
**url(string): url for notification button. Open a link when the button is pressed. (optional)*
**color(string): color for notification button. Accept hexadecimal colors, like: '#bcbcbc'. (optional)*

Returns
(Promise)

```
import tago

my_account = tago.Account('54d83222-6837-4e9e-8f2e-67de8fce5a8b')

data = {
    'title': 'Temperature Alert',
    'message': 'The temperature is too high'
    'buttons': [{
        'label': 'Go to device dashboard',
        'url': 'https://admin.tago.io/dashboard/info/5d9c6e7945f7ab001b0a32c2',
        'color': 'red',
        # 'analysis': '5d9c6e7945f7ab001b0a32c2',
    }],
}

result = my_account.run.notificationEdit('NOTIFICATION_ID_HERE', data)
print(result)
```

3.13.3 .notificationCreate

Create a new push notification for the user.

Syntax
.notificationCreate(/data/)

Arguments

*user_id(string): ID of the run user;
data(object) options to be modified in the notification.
*title(string): a title for the notification.
*message(string): message for the notification.
*buttons(array of object): phone for the run user.
*label(string): label for notification button.
*analysis(string): analysis_id for notification button. This analysis is run when the button is pressed. (optional)
*url(string): url for notification button. Open a link when the button is pressed. (optional)
*color(string): color for notification button. Accept hexadecimal colors, like: '#bcbcbc'. (optional)

Returns
(Promise)

```
import tago

my_account = tago.Account('54d83222-6837-4e9e-8f2e-67de8fce5a8b')

data = {
    'title': 'Temperature Alert',
    'message': 'The temperature is too high',
    'buttons': [{
        'label': 'Go to device dashboard',
        'url': 'https://admin.tago.io/dashboard/info/5d9c6e7945f7ab001b0a32c2',
        'color': 'red',
        # 'analysis': '5d9c6e7945f7ab001b0a32c2',
    }],
}

result = my_account.run.notificationCreate('RUN_USER_ID_HERE', data)
print(result)
```

3.13.4 .notificationDelete

Delete push notifcation for the run user

Syntax
.notificationDelete()

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

Returns
(Promise)

```
import tago

my_account = tago.Account('54d83222-6837-4e9e-8f2e-67de8fce5a8b')

result = my_account.run.notificationDelete('NOTIFICATION_ID_HERE')
print(result)
```

3.14 Access Management

Access Management (AM) is a module that helps you securely grant access to certain resources in your account. You create Targets (users or things) and determine which type of Permissions for the resources they will have. See more about Access Management here.

3.14.1 .list

Retrieve a list with all access management from account.

Syntax
.list()

Returns
(Promise)

```
import tago

my_account = tago.Account('54d83222-6837-4e9e-8f2e-67de8fce5a8b')

access_management_list = my_account.accessManagement.list()
print(access_management_list)
```

3.14.2 .create

Generate and retrieve a new access management for the account.

Syntax
.create(/data/)

Arguments
data(object) options for the new access management.
**name(string): a name for the access management.*
**permissions(array): permissions for the access management.*
**effect(string): effect for the access management. access or deny.*
**action(string): action for the access management.*
**resource(string): resource for the access management.*

targets(array): *targets for the access management.*
active(bool): *Set if the access management will be visible. Default True. (optional)*
tags(array): *An array of objects with key and value. (optional)*

Returns
(Promise)
am_id: *id of the new access management;*

```
import tago

my_account = tago.Account('54d83222-6837-4e9e-8f2e-67de8fce5a8b')

user = {
    'id': '576dc932415f403531fd2cf6',
    'name': 'John Doe',
}

new_access_management = {
    'name': 'Dashboards for the user {}'.format(user['id']),
    'tags': [{ 'key': 'client_id', 'value': user['id'] }],
    'targets': [
    [
        'run_user',
        'id',
        user['id'],
    ],
    ],
    'permissions': [
    {
        'effect': 'allow',
        'action': [
        'access',
        ],
        'resource': [
        'dashboard',
        'tag.key',
        'client_id',
        'tag.value',
        user['id'],
        ],
    },
    ],
}

result = my_account.accessManagement.create(new_access_management)
print(result)
```

3.14.3 .edit

Modify any property of the access management.

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

Arguments
am_id(string) reference ID of the access management.
data(object) options for the new access management.
**name(string): a name for the access management.(optional)*
**permissions(array of object): permissions for the access management.(optional)*
**effect(string): effect for the access management. access or deny (optional)*
**action(string): action for the access management.(optional)*
**resource(string): resource for the access management.(optional)*
**targets(array of arrays): targets for the access management.(optional)*
**active(bool): Set if the access management will be visible. Default True. (optional)*
**tags(array): An array of objects with key and value. (optional)*

Returns
(Promise)

```
import tago

my_account = tago.Account('54d83222-6837-4e9e-8f2e-67de8fce5a8b')

data = {
    'name': 'my new name of access management',
}

result = my_account.accessManagement.edit('ACCESS_MANAGEMENT_ID', data)
print(result)
```

3.14.4 .info

Get information about the access management

Syntax
.info(/id/)

Arguments
id(string) reference ID of the access management.

Returns
(Promise)

```
import tago

my_account = tago.Account('54d83222-6837-4e9e-8f2e-67de8fce5a8b')

access_management_info = my_account.accessManagement.info('5dc0b9ab9955dd0026247ce6')
print(access_management_info)
```

3.14.5 .delete

Delete access management for the account

Syntax
.delete(/id/)

Arguments
id(string) reference ID of the access management.

Returns
(Promise)

```
import tago

my_account = tago.Account('54d83222-6837-4e9e-8f2e-67de8fce5a8b')

result = my_account.accessManagement.delete('5dc0b9ab9955dd0026247ce6')
print(result)
```