tago-documentation Documentation

Release latest

Tago LLC

Mar 30, 2017

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.

.info

Get all information from the device

Syntax
.info()

Returns

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Device device = new Device("7c16da11-2101-4ea3-9568-7249115d73f3");

Result res = device.info("58d5318eabd6a6000e542b95");
```

.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

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Device device = new Device("8aa46f99-3156-4ebd-a275-fdb75c4dccbf");

final Object loc = new Object() {
    public Double lat = 42.2974279;
    public Double lng = -85.628292;
};

Object data = new Object() {
    public String variable = "temperature";
    public String unit = "F";
    public String value = "55";
    public String time = "2015-11-03 13:44:33";
    public Object location = loc;
};

Result res = device.insert(data);
```

.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

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Device device = new Device("8aa46f99-3156-4ebd-a275-fdb75c4dccbf");

Object filter = new Object(){
    public String variable = "myvar";
    public String query = "last_value";
    public String end_date = "2014-12-25 23:33:22";
    public String start_date = "2014-12-20 23:33:22";
};

Result res = device.find(filter);
```

.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

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Device device = new Device("8aa46f99-3156-4ebd-a275-fdb75c4dccbf");

Object filter = new Object(){
    public String variable = "myvar";
    public String query = "last_value";
    public String end_date = "2014-12-25 23:33:22";
    public String start_date = "2014-12-20 23:33:22";
};

Result res = device.remove(null, null);
```

or

```
Device device = new Device("8aa46f99-3156-4ebd-a275-fdb75c4dccbf");

Object filter = new Object(){
    public String variable = "myvar";
    public String query = "last_value";
    public String end_date = "2014-12-25 23:33:22";
    public String start_date = "2014-12-20 23:33:22";
};

Result res = device.remove("myvar", null);
```

or

```
Device device = new Device("8aa46f99-3156-4ebd-a275-fdb75c4dccbf");

Object filter = new Object(){
    public String variable = "myvar";
    public String query = "last_value";
    public String end_date = "2014-12-25 23:33:22";
    public String start_date = "2014-12-20 23:33:22";
};

Result res = device.remove("577d81ac7ee399ef1a6e98da", 1);
```

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
.listening(listener, "analysis token")

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

```
Analysis myAnalysis = new Analysis();

Listener listener = new Listener() {
    @Override
    public void call(Object object, Console console) {
        System.out.println("this logs the local console");
        console.log("this logs the tago analysis console");
    }
```

```
};

myAnalysis.listening(listener, "d43b1695-d8a8-44f5-ae8b-512a7ecffdb9");
```

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:

myanalysis.listening(listener, "d43b1695-d8a8-44f5-ae8b-512a7ecffdb9")

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.

```
Analysis myanalysis = new Analysis("d43b1695-d8a8-44f5-ae8b-512a7ecffdb9");

myanalysis.sms.send(data);
```

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

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Analysis myanalysis = new Analysis("d43b1695-d8a8-44f5-ae8b-512a7ecffdb9");

Object data = new Object() {
    public String to = "2693856214";
    public String message = "'Hi! This is a sms example sent from Tago. \\nWith a␣
↪breakline in the sms message.";
};
Result res = myanalysis.sms.send(data);
```

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/)

Arguments
to(string) E-mail address which will receive the email.
subject(string) Subject of the email;
message(string) message to be sent.
from(string) E-mail address for the receiver to reply. Default is tago@tago.io (optional);

Returns

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Analysis myanalysis = new Analysis("d43b1695-d8a8-44f5-ae8b-512a7ecffdb9");

Object data = new Object() {
    public String to = "myuser@gmail.com";
    public String subject = "E-mail example";
    public String message = "Hi! This is an email example.";
    public String to = "me@gmail.com";
};
Result res = myanalysis.email.send(data);
```

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.

.info

Get all account information

Syntax
.info()

Returns

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account account = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

Result res = myacc.info();
```

.tokenList

Get all tokens from the account

Syntax
.tokenList()

Returns

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account account = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

Result res = myacc.tokenList();
```

.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;*
**password(string): password of the account;*
**expire_time(string): Time when token should expire. It will be randomly generated if not included.*

Returns

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

Object data = new Object(){
    public String name = "My First Token";
    public String expire_time = "never";
```

```
    public String password = "pass";
};
Result res = myacc.tokenCreate(data);
```

.tokenDelete

Delete current token of the account

Syntax
.tokenDelete()

Returns

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

Result res = myacc.tokenDelete();
```

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

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

Result res = myacc.device.list();
```

.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

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

final List<Object> confParams = new ArrayList<>();

confParams.add(new Object(){
    public Boolean sent = false;
    public String key = "check_rate";
    public String value = "600";
});

    confParams.add(new Object(){
    public Boolean sent = false;
    public String key = "measure_time";
    public String value = "0";
});

final List<Object> tagParams = new ArrayList<>();
tagParams.add(new Object(){
    public String key = "client";
    public String value = "John";
});
```

```
Object data = new Object(){
    public String name = "My first device";
    public String description = "Creating my first device";
    public Boolean active = true;
    public Boolean visible = true;
    public List<Object> configuration_params = confParams;
    public List<Object> tags = tagParams;
};

Result res = myacc.device.create(data);
```

.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

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

final List<Object> tagParams = new ArrayList<>();
tagParams.add(new Object(){
    public String key = "client";
    public String value = "Mark";
});

Object data = new Object(){
    public String name = "New name for my device";
    public String description = "In this way I can change the description too";
    public Boolean active = false;
    public Boolean visible = true;
    public List<Object> tags = tagParams;
};
```

```
Result res = myacc.device.edit("58da9eac20c52d000e786748", data);
```

.info

Get information about the device

Syntax
.info(/id/)

Arguments
id(string) reference ID of the device.

Returns

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

Result res = myacc.device.info("58da9eac20c52d000e786748");
```

.delete

Delete device for the account

Syntax
.delete(/id/)

Arguments
id(string) reference ID of the device.

Returns

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

Result res = myacc.device.delete("58da9eac20c52d000e786748");
```

.tokenList

Retrieve a list of all tokens of the device

Syntax
.tokenList(/id/)

Arguments
id(string) reference ID of the device.

Returns

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

Result res = myacc.device.tokenList("58da9eac20c52d000e786748");
```

.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'.*

Returns

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

Object data = new Object(){
    public String name = "My First Token";
    public String expire_time = "never";
    public String permission = "full";

};

Result res = myacc.device.tokenCreate("58daa3c44cd1310033b4fcaf", data);
```

.tokenDelete

Delete an token of the Device

Syntax
.tokenDelete(/token/)

Arguments
token(string) reference token.

Returns

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

Result res = myacc.device.tokenDelete("a021a360-21ab-4318-87c0-6cd584a20a3f");
```

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

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

 Result res = myacc.bucket.list();
```

.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

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

final List<Object> tagParams = new ArrayList<>();
tagParams.add(new Object(){
    public String key = "client";
    public String value = "Francisco";
});

Object data = new Object(){
    public String name = "My first bucket";
    public String description = "Creating my first bucket";
```

```
    public Boolean visible = true;
    public List<Object> tags = tagParams;
};

Result res = myacc.bucket.create(data);
```

.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

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

final List<Object> tagParams = new ArrayList<>();
tagParams.add(new Object() {
    public String key = "client";
    public String value = "Leonardo";
});

Object data = new Object() {
    public String name = "New name for my bucket";
    public String description = "This way I can change the description too";
    public Boolean visible = true;
    public List<Object> tags = tagParams;
};

Result res = myacc.bucket.edit("58daaac929d6e4000ee13d0e", data);
```

.info

Get information about the bucket

Syntax
.info(/id/)

Arguments
id(string) reference ID of the bucket.

Returns

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

Result res = myacc.bucket.info("58daaac929d6e4000ee13d0e");
```

.delete

Delete bucket for the account

Syntax
.delete(/id/)

Arguments
id(string) reference ID of the bucket.

Returns

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

Result res = myacc.bucket.delete("58daaac929d6e4000ee13d0e");
```

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

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

Result res = myacc.action.list();
```

.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

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

final List<Object> tagParams = new ArrayList<>();
tagParams.add(new Object() {
    public String key = "Trigger";
    public String value = "2";
});

Object data = new Object() {
    public String name = "a simple action";
    public String description = "trigger when the variable test is higher than 2, and␣
↪reset it when is less than 2";
    public String when_reset_bucket = "571920982c452fa00c6af660";
    public String when_reset_origin = "571920a5cc7d43a00c642ca1";
    public String when_reset_variable = "test";
    public String when_reset_condition = "<";
    public String when_reset_value = "2";
    public String when_set_bucket = "571920982c452fa00c6af660";
    public String when_set_origin = "571920a5cc7d43a00c642ca1";
    public String when_set_variable = "test";
    public String when_set_condition = ">";
    public String when_set_value = "2";
    public String type = "script";
    public String script = "577d4c457ee399ef1a6e6cf6";
    public Boolean lock = false;
    public Boolean active = true;
    public List<Object> tags = tagParams;
};

Result res = myacc.action.create(data);
```

.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

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

final List<Object> tagParams = new ArrayList<>();
tagParams.add(new Object() {
    public String key = "client";
    public String value = "Mark";
});

Object data = new Object() {
    public String name = "New name for my action";
    public String description = "In this way I can change the description too";
    public Boolean visible = true;
    public List<Object> tags = tagParams;
};

Result res = myacc.action.edit("58daafb04cd1310033b516e2", data);
```

.info

Get information about the action

Syntax
.info(/id/)

Arguments
id(string) reference ID of the action.

Returns

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

Result res = myacc.action.info("58daafb04cd1310033b516e2");
```

.delete

Delete action for the account

Syntax
.delete(/id/)

Arguments
id(string) reference ID of the action.

Returns

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

Result res = myacc.action.delete("58daafb04cd1310033b516e2");
```

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

```
Result(){
  public Boolean status;
  public String message;
```

```
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

Result res = myacc.analysis.list();
```

.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

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

final List<Object> varParams = new ArrayList<>();
varParams.add(new Object() {
    public String key = "max_battery";
    public String value = "3100";
});

final List<Object> tagParams = new ArrayList<>();
tagParams.add(new Object() {
    public String key = "client";
    public String value = "Mark";
});

Object data = new Object() {
    public String name = "My first analysis";
    public String description = "Creating my first analysis";
    public Boolean active = true;
    public String interval = "1 minute";
```

```
    public List<Object> variables = varParams;
    public List<Object> tags = tagParams;
};

Result res = myacc.analysis.create(data);
```

.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

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

final List<Object> varParams = new ArrayList<>();
varParams.add(new Object() {
    public String key = "max_battery";
    public String value = "3000";
});

final List<Object> tagParams = new ArrayList<>();
tagParams.add(new Object() {
    public String key = "client";
    public String value = "Mark";
});

Object data = new Object() {
    public String name = "New name for my analysis";
    public String description = "In this way I can change the description too";
    public Boolean active = false;
    public String interval = "2 minutes";
```

```
    public List<Object> variables = varParams;
    public List<Object> tags = tagParams;
};

Result res = myacc.analysis.edit("58d406eae69ebf000e6edfed", data);
```

.info

Get information about the analysis

Syntax
.info(/id/)

Arguments
id(string) reference ID of the analysis.

Returns

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

Result res = myacc.analysis.info("58d406eae69ebf000e6edfed");
```

.delete

Delete analysis for the account

Syntax
.delete(/id/)

Arguments
id(string) reference ID of the analysis.

Returns

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

Result res = myacc.analysis.delete("58d406eae69ebf000e6edfed");
```

.run

Force Analysis to run immediately

Syntax
.run(/id/)

Arguments
id(string) reference ID of the analysis.

Returns

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

Object scope = new Object(){

};

Result res = myacc.analysis.run("58d406eae69ebf000e6edfed", scope);
```

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

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

Result res = myacc.dashboard.list();
```

.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

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
 Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

final List<Object> arrParams = new ArrayList<>();

 arrParams.add(new Object() {
     public String widget_id = "577c28d269d2861f1b2e93b8";
     public Integer x = 0;
     public Integer y = 0;
     public Integer width = 2;
     public Integer height = 3;
 });

 final List<Object> tagParams = new ArrayList<>();
```

```
tagParams.add(new Object() {
    public String key = "client";
    public String value = "Mark";
});

Object data = new Object() {
    public String label = "My first dashboard";
    public List<Object> arrangement = arrParams;
    public List<Object> tags = tagParams;
};

Result res = myacc.dashboard.create(data);
```

.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

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

Object data = new Object() {
    public String label = "New name for my dashboard";
};


Result res = myacc.dashboard.edit("58dac53e20c52d000e78b4d2", data);
```

.info

Get information about the dashboards

Syntax
.info(/id/)

Arguments
id(string) reference ID of the dashboards.

Returns

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

Result res = myacc.dashboard.info("58dac53e20c52d000e78b4d2");
```

.delete

Delete dashboards for the account

Syntax
.delete(/id/)

Arguments
id(string) reference ID of the dashboards.

Returns

```
Result(){
  public Boolean status;
  public String message;
  public Object result;
}
```

```
Account myacc = new Account("7c16da11-2101-4ea3-9568-7249115d73f3");

Result res = myacc.dashboard.delete("58dac53e20c52d000e78b4d2");
```

#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