Digitaleo

BUSINESS BOOSTER

Voice API

Révisions

Revision	Author	Date	Comments
1	pmarechal	01/02/2014	Version initiale de la documentation
2	pmarechal	03/08/2015	Authentification oAuth 2.0
3	pmarechal	11/01/2015	Ajout du lien vers la bibliothèque PHP facilitant l'intégration
4	pmarechal	19/02/2016	English version
5	athomas	29/04/2019	Add "type" parameter for "VOICE" media

Table of contents

This document describes the programming interface, or API, of Digitaleo's platform for sending voice messages.

Voice messages can transit via the two different channels:

- The PUSH voice channel. A call is made to the destination number (mobile telephone or fixed telephone) and the message is broadcast when the person answers the phone;
- The Message on answering machine channel. As its name indicates, the voice message is deposited directly into the recipient's mobile telephone answering machine.

The API can be broken down into three steps

- Importing your audio file to our platform;
- Sending the audio file to the various recipients;
- Retrieving the status of the sending to each recipient.

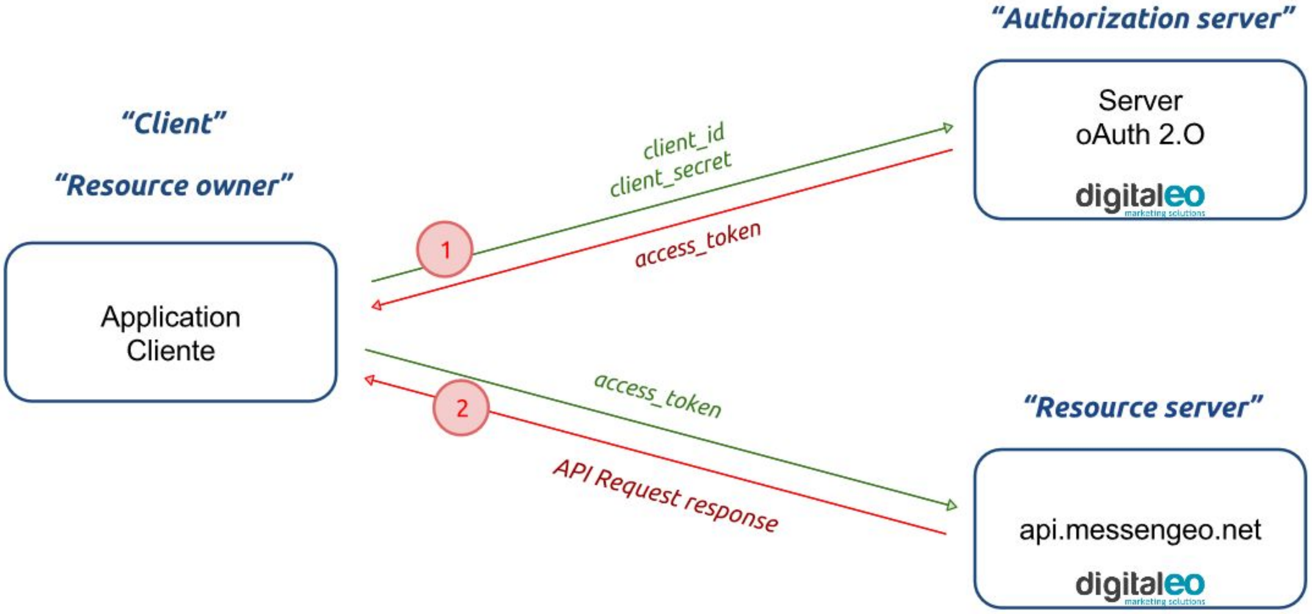
1. Overview

This API is not RESTfull because for all of the calls, the verbs HTTP GET and POST can be used. However, it is based on various resources of which the details are provided further on in this document.

The purpose of this first section is to help you understand the various types of calls to our APIs, regardless of the resource.

1.1. Authentification

Authentication to our APIs is based on the oAuth 2.0 protocol. Each call to our APIs has to contain an access_token that the client application will have requested beforehand from the Digitaleo authorization server:



1.1.1. Retrieving the application ids

To retrieve a client_id and a client_secret, you must declare an application in the Digitaleo platform.
For this,

1. Connect to app.digitaleo.com
2. Click on the Parameters menu
3. Go to the API tab

1.1.2. Retrieve an authentication token

The client must perform a POST request with the following parameters:

- grant_type: The value must be "client_credentials" for this type of authorization
- client_id: The id of the application (client)
- client_secret: The secret key of the application (client)

Note: The client_id and client_secret will be sent to you.

The URL for retrieving a token is as follows

```
https://oauth.messengeo.net/token
```

Example of an HTTP request

```
POST /token HTTP/1.1
Host: oauth.messengeo.net
Content-Type: application/x-www-form-urlencoded

client_id=51612c780b4dbaea8f81995beccbcfec08969d0e&
client_secret=p280edbd76d510c41990cbe5e6108c7e&
grant_type=client_credentials
```

Example of a request with Curl

```
curl https://oauth.messengeo.net/token
  -d 'client_id=51612c780b4dbaea8f81995beccbcfec08969d0e'
  -d 'client_secret=p280edbd76d510c41990cbe5e6108c7e'
  -d 'grant_type=client_credentials'
```

Return

if successful, the authorization server will return a code 200 HTTP response of which the body will contain the following JSON flow

```
{
   "access_token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpZCI6ImYyMzE2...",
   "expires_in":"3600",
   "token_type":"bearer",
   "scope":"basic",
}
```

Description of the various fields:

Property	Description
access_token	The token issued by the authorization server. *Note: The size of the token can range up to 50,000 characters*
expires_in	The lifespan in seconds of the token issued
token_type	The type of token. The Digitaleo server only supports the "bearer" type
scope	The scope of the token

If one of the parameters is not correct, the authorization server will return a code 400 http response (HTTP/1.1 400 Bad Request) of which the body will contain the following json flow:

```
{
   "error":"invalid_client",
   "error_description":"The client credentials are invalid",
}
```

1.1.3. Using the authorization token (access_token)

The authorization token is sent to the API in the header of the HTTP request and more particularly in the header “Authorization: Bearer”. Note that the “Authorization: Bearer” is case-sensitive.

Example of an HTTP request

```
GET /rest/campaigns HTTP/1.1

Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpZCI6ImYyMzE2…
Host: api.messengeo.net
```

Example of a request with Curl

```
curl -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpZCI6ImYyMzE2..."
https://api.messengeo.net/rest/campaigns
```

1.2. Pagination, sorting and filtering

1.2.1. Pagination

1.2.1.1. introduction

Three parameters allow you to manage the paginated resource display.

The parameters are

- **limit**: allows you to limit the number of elements returned
- **offset**: allows you to ignore the first n elements of the list
- **total**: allows you to retrieve the total number of resources if indeed the request had not limited the number of resources returned. It is therefore useful in the framework of using a limit for pagination. (the default is false). This feature uses a lot of resources. It is recommended that it not be activated in the case there is no pagination.

1.2.1.2. Example 1: Limiting results

Retrieving only 20 resources and the total number of resources if the result were limited to 20 resources

```
GET /rest/ressource HTTP/1.1
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpZCI6ImYyMzE2…
Host: api.messengeo.net
Content-Type: application/x-www-form-urlencoded

limit=20&total=true
```

with Curl

```
curl
 -X GET
 -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbG...ccOqbVow8xOQyQ"
 -d limit=20
 -d total=true
https://api.messengeo.net/rest/ressource
```

1.2.1.3. Example 2: Pagination

Retrieving 10 resources, leaving aside the first 20. This boils down to reading the 3rd page knowing that each page lists 10 resources.

```
GET /rest/resource HTTP/1.1
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpZCI6ImYyMzE2…
Host: api.messengeo.net
Content-Type: application/x-www-form-urlencoded

limit=10&offset=20
```

1.2.2. Sorting

1.2.2.1. Introduction

The **sort** parameter allows you to order the list of resources returned according to one of the attributes of the resource concerned. This parameter is comprised of two elements separated by a space:

- The name of the attribute according to which you want to order the list
- The sorting order, ascending order (ASC) or descending order (DESC)

1.2.2.2. Exemple

Sorting a list of resources in descending order according to the id of this resource:

```
GET /rest/resource HTTP/1.1
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpZCI6ImYyMzE2…
Host: api.messengeo.net
Content-Type: application/x-www-form-urlencoded

sort=id%20DESC
```

with Curl

```
curl
 -X GET
 -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbG...ccOqbVow8xOQyQ"
 -d sort=id DESC
https://api.messengeo.net/rest/ressource
```

1.2.3. Limiting the list of attributes returned per resource

1.2.3.1. Introduction

It is possible to limit the list of attributes of the resources returned. In other words, this entails returning incomplete resources in order to focus on the attributes that are really necessary for the client that generated the call.

This makes it possible to save both bandwidth and processing on the server side. Indeed, certain attributes are calculated at the time of the call and not retrieving them makes it possible to reduce the request time.

The parameter that allows you to define the attributed return is called properties.

For each resource, a list of attributes returned by default (if the properties parameter is not defined) is imposed. An alias called DEFAULT makes it possible to specify that you want to retrieve the attributes by default + another or – another.

1.2.3.2. Example 1

Retrieving only the is and the name of each resource

```
GET /rest/resource HTTP/1.1
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpZCI6ImYyMzE2…
Host: api.messengeo.net
Content-Type: application/x-www-form-urlencoded

properties=id,name
```

with Curl

```
curl
 -X GET
 -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbG...ccOqbVow8xOQyQ"
 -d properties=id,name
https://api.messengeo.net/rest/ressource
```

1.2.3.3. Example 2

Retrieving the attributes returned by default except the id

```
GET /rest/resource HTTP/1.1
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpZCI6ImYyMzE2…
Host: api.messengeo.net
Content-Type: application/x-www-form-urlencoded

properties=DEFAULT,-id
```

with Curl

```
curl
 -X GET
 -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbG...ccOqbVow8xOQyQ"
 -d properties=DEFAULT,-id
https://api.messengeo.net/rest/ressource
```

1.3. The lists of resources returned

Most of the calls to the REST API return a list of resources. This list of resources is comprised of the following three elements:

- **size**: The number of resources returned
- **total**: The number of resources returned if the request had not limited the result
- **list**: The table containing the resources

An example of a list of resources returned in json format

```
{
    "size": 2,        // The number of resources returned
    "total": 600640, // The number of resources returned if the request had not limited the result
    "list":           //  The table containing the resources
    [
    {
        "id": "1",
        "email": "aladdin@digitaleo.com",
        "phone": "+33201010101",
        "mobile": "+33601010101",
    },
    {
        "id": "2",
        "email": "jasmine@digitaleo.com",
        "phone": "+33202020202",
        "mobile": "+33602020202",
    }
    ]
}
```

1.4. . The various actions on a resource

1.4.1. Introduction

Our APIs comply with the HTTP verbs and their correspondence with the CRUD actions (Create, Read, Update, Delete) of a resource. However, it is also possible to perform all of the actions only with HTTP GET requests or only with HTTP POST requests. To do this, the action to be performed must be specified in the URL.

Description of the action	Dedicated HTTP verb	Name of the action
Read resources	GET	read
Create a resource	POST	create
Update resources	PUT	update
Delete resources	DELETE	delete

1.4.2. Example

The two following Curl requests are considered to be equivalent by our APIs

```
curl
 -X GET
 -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbG...ccOqbVow8xOQyQ"
https://api.messengeo.net/rest/ressource
```

```
curl
 -X POST
 -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbG...ccOqbVow8xOQyQ"
https://api.messengeo.net/rest/ressource?action=read
```

1.4.3. Profil des méthodes en fonction des actions

Description of the action	input	output
read	Filter	List of resources
create	Parameters	Resource created
update	Filter + Parameter	Number of resources updated
delete	Filter	Number of resources deleted

1.5. Updating resources

Updating resources is based on two types of parameters:

- A list of filters that allows you to limit the resources to be updated
- A list of keys/values that allow you to define which resource attributes to update, with which value.

The keys/values to be updated must be grouped together in a "metaData" attribute, in the form of a JSON flow.

For example, the following request makes it possible to update the name and the description of resources for which the is is either 100, or 200 and for which the name is equal to "former name of the resource"

```
PUT /rest/ressource HTTP/1.1
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpZCI6ImYyMzE2…
Host: api.messengeo.net
Content-Type: application/x-www-form-urlencoded

id=100,200&name='ancien nom de la ressource'&metaData='{"name":"Le nouveau nom de la
ressource","description":"La nouvelle description de la ressource"}'
```

with Curl

```
curl
 -X PUT
 -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbG...ccOqbVow8xOQyQ"
 -d id=100,200
 -d name='ancien nom de la ressource'
 -d metaData='{"name":"Le nouveau nom de la ressource","description":"La nouvelle
description de la ressource"}'
 https://api.messengeo.net/rest/ressource
```

Actions that update resources return the number of contacts involved by the filter passed as input

```
{
   count: 10,
}
```

1.6. Return codes

The HTTP response code is contained:

- in the HTTP header,
- in the content of the response in the case of an error.

The return codes are based on the HTTP return codes:

- 2XX - The call to the API unfolded correctly
- 4XX – The call to the API has an error in its parameters.

Codes with success:

- **200 OK**: everything went well
- **201 Created**: Resource created
- **204 No Content**: Resource updated or deleted

The error codes that you are likely to see are the following:

- **304 Not Modified**: Error during updating or deleting (the resource is not modified)
- **400 Bad Request**: Missing or incorrect parameter
- **401 Unauthorized**: Authentication failed
- **403 Forbidden**: Access to the requested location is prohibited
- **404 Not Found**: Unknown method or method not indicated
- **405 Method Not Allowed**: You are not authorized to use the method that you are requesting
- **414 Request-URI Too Long**: Your request is too large, please shorten it
- **417 Expectation Failed**: The required parameters are either missing or are incorrect
- **500 Internal Server Error**: Unidentified error

For example, if the authentication token is no longer valid for the following request:

```
curl
 -X GET
 -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbG...ccOqbVow8xOQyQ"
https://api.messengeo.net/rest/ressource
```

The header of the HTTP response will be

```
< HTTP/1.1 401 Unauthorized
< Date: Fri, 06 Mar 2015 21:32:06 GMT
< Server: Apache/2.2.16 (Debian)
< X-Powered-By: PHP/5.3.3-7+squeeze15
< Expires: Thu, 19 Nov 1981 08:52:00 GMT
< Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
< Pragma: no-cache
< Content-Length: 46
< Content-Type: application/json
```

while the body of the HTTP response will be

```
{
    "status": 401,
    "message": "Authenticate failed"
}
```

1.7. Response formats

1.7.1. Introduction

The REST API can respond to the requests in different formats. By default, it returns a response in JSON format but it can also return a response in XML, CSV (for certain resources) and JS (JSONP) formats.

To change the format, .xml, .json, .csv or .js must be added to the end of the URI regardless of the HTTP verb (GET, POST, DELETE or PUT)

1.7.2. Examples

To retrieve the list of mailings in JSON format

```
GET /rest/ressource.json HTTP/1.1
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpZCI6ImYyMzE2…
Host: api.messengeo.net
Content-Type: application/x-www-form-urlencoded
```

To retrieve the list of mailings in XML format

```
GET /rest/ressource.xml HTTP/1.1
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpZCI6ImYyMzE2…
Host: api.messengeo.net
Content-Type: application/x-www-form-urlencoded
```

To retrieve the list of mailings in CSV format

```
GET /rest/ressource.csv HTTP/1.1
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpZCI6ImYyMzE2…
Host: api.messengeo.net
Content-Type: application/x-www-form-urlencoded
```

To retrieve the list of mailings in JSONP format

```
GET /rest/ressource.js HTTP/1.1
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpZCI6ImYyMzE2…
Host: api.messengeo.net
Content-Type: application/x-www-form-urlencoded

callback=yourFunctionCallback
```

1.7.3. Gestion du cross-domain

Retrieving the response in javascript format allows you to overcome the difficulties linked to the cross-domain. Passing through a server in order to consult the API's directly is thus avoided. On the client side, it is recommended to use the jquery-jsonp plugin (jQuery-jsonp on GitHub) for error management (not initially available in JQuery).

For example, reading the resource of which the id is 2 via an ajax request returns

```
<script type="text/javascript" language="javascript" src="jquery.jsonp.js"></script>
<script>
  $.jsonp({
      url: 'https://api.messengeo.net/rest/ressource.js?callback=?',
      beforeSend: function (request) {
          request.setRequestHeader("Authorization", "Bearer " + ($("#accesstoken").val()));
      },
      data: {
          id: '2',
      }
  }).done(function(data) {
     // data peut être un objectlist (size, total, list) ou une erreur (status, message)
     console.log(data);
  }).fail(function(jqxhr, textStatus, errorThrown) {
     console.log('Errors occured');
  });
</script>
```

1.8. Filters and passing multiple values

The read, update and delete actions use as input a filter which makes it possible to select only the resources to be read or to be affected. Most of these filters take several values.

There are two ways to pass these multiple values:

1. in the form of a character string with the values separated by commas;
2. in the form of a table.

For example, the following two requests allow you to retrieve the resources for which the is is equal either to 12, or equal to 13.

```
GET /rest/ressource.json HTTP/1.1
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpZCI6ImYyMzE2…
Host: api.messengeo.net
Content-Type: application/x-www-form-urlencoded

id=12,13
```

```
GET /rest/ressource.json HTTP/1.1
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpZCI6ImYyMzE2…
Host: api.messengeo.net
Content-Type: application/x-www-form-urlencoded

id[0]=12&id[1]=13
```

1.9. Integrating our API as PHP

In order to simplify the integration of our REST APIs, we provide you with a library that facilitates the various calls from code written in php. This library is hosted on GitHub.

2. Importing an audio file on the Digitaleo platform

Before submitting a voice campaign, the mp3 or wav file has to be imported on the Digitaleo platform. Digitaleo provides a content management API that allows you to store files and to broadcast them via https. Once present on the Digitaleo platform, it is converted to PCM format, 8kHz, Mono, 16 bit resolution, encoding A--Law 8 bits.

This is the format that the recipient will hear. In order to ensure the quality of the audio file, it is possible to download the PCM format thanks to the link present in the url_encoded attribute of the resource returned following import.

The uploaded audio file must not exceed 20Mb in size and must last at least 10sec.

2.1. List of parameters to supply in order to create audio content

Parameter	Type	Description
name *required*	string	Name of the audio content
contentFile *required*	binary	Binary content of the file to be stored. *It must not exceed 20Mb in size.*
description *optional*	string	Description of the audio content
tags *optional*	string	table of keywords to associate with the audio content. This makes it possible to search for the list of audio content that corresponds to a particular keyword

2.2. Example of creating audio content

```
POST /content/rest/contents HTTP/1.1
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpZCI6ImYyMzE2…
Host:baseo.messengeo.net
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW

----WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="name"

Mon enreegistrement vocal
----WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="contentFile"; filename="monEnregistrementVocal.mp3"
Content-Type: audio/mp3

----WebKitFormBoundary7MA4YWxkTrZu0gW
```

Note: The content-type for the request must be "multipart/form-data" in order to upload the file.

and with Curl

```
curl
 -X POST
 -H 'Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpZCI6ImYyMzE2…'
 -F "name=Mon enregistrement vocal"
 -F "contentFile=@/home/pierre/monEnregistrementVocal.mp3"
 https://baseo.messengeo.net/content/rest/contents
```

2.3. Return

```
{
   "size": 1,
   "total": null,
   “list": [
   {
     "id": "320442",
     "name": "Mon enregistrement vocal",
     "description": null,
     "comment": null,
     "filename": "monEnregistrementVocal.mp3",
     "filesize": "131821",
     "mimeType": "audio/mp3",
     "duration":15,
     "tags": null,
     “url": "https://datas.messengeo.net/22ec...f702a/monEnregistrementVocal.mp3",
     "url_encoded":"https://datas.messengeo.net/22ec...f702a/monEnregistrementVocal.pcm",
     …
  }
  ]
}
```

3. Creating the voice mailing

3.1. List of parameters to supply in order to create a voice mailing

Parameter	Description
name *optional*	Name of the mailing. The name allows you to subsequently retrieve the information linked to this mailing.
link *required*	Link representing the content: ● VOICE: link to a VXML, WAV or MP3 file; ● VOICEMAIL: link to a PCM file.
media *required*	Media via which to broadcast the audio message: ● VOICE if it involves PUSH voice ● VOICEMAIL if it involves a message on an answering machine
type *optional*	Required if the media is “VOICE”. - “vxml”, “wav” (choose “wav” for WAV and MP3 files)
date *optional*	Sending date of the voice mailing in ISO-8601 format. If no date is specified or if the date is in the past, the mailing will be sent immediately.

contacts
required
	The list of contacts (telephone numbers) for this campaign.
	This field must have the following structure:
	`[`
	`{`
	`“recipient”:”+33612345678”,`
	`},`
	`{`
	`“recipient”:”+33612345679”,`
	`},`
	`{`
	`“recipient”:”+33612345680”,`
	`},`
	`]`

pingUrl *optional*	Notification url for a status change on one or more messages.

3.2. Example of creating a Message on answering machine mailing

```
curl
--F "name=Campagne Vocale Soldes 2014"
--F "link=https://datas.messengeo.net/7af6f60240aa6d9c7a5009b86b200a7d/20dol.pcm"
--F "media=VOICEMAIL"
--F "date=2014--02--12T15:00:00+01:00"
--F
"contacts=[{"recipient":"+33612345678"},{"recipient":"+33612345679"},{"recipient":"+33612345
680"}]"
https://api.messengeo.net/rest/mailings
```

3.3. Telephone numbers accepted

For sendings to the VOICE *(Push voice)* or VOICEMAIL *(Message on answering machine)* media, the telephone numbers must comply with one of the following formats:

- International format: +33 6 xx xx xx xx, *for example: +33 6 12 34 56 78*
- Local format: 06 xx xx xx xx, *for example: 06 12 34 56 78*

Note that the following format is not supported: 00336 xx xx xx xx

The international format is the recommended format

numbers in local format are automatically transformed into international numbers by the Digitaleo platform based on the default indicator for your account.

Support for mobiles and fixed telephones according to the media

Media	Mobile	Fixed
VOICE *(Push vocal)*	yes	yes
VOICEMAIL *(Message on answering machine)*	yes	no

3.4. Limit in the number of contacts per mailing creation

Creating an emailing is currently limited to 1000 contacts. However, when reading statistics, you can consolidate the statistics of several mailings that have the same name.

3.5. Specifying a URL or PING

Instead of regularly polling the status of a sent message until it is received ("pull" mode), you can be informed ("push" mode) of the arrival of a status linked to the messages that you have sent.

For this, you supply, when the mailing is created, an address on your server ‘http://monserveur.com/ping.php?id=#id#’ for example, and the Digitaleo platform will call this address with the id of the email when its status changes.

This URL is passed in the pingUrl parameter.

Cumulating changes in status ("batch") is possible by using the #ids# pattern instead of #id#. The ids are then separated by commas.

3.6. Complete example (Importing the file and sending)

```php
<?php

// Le code source de la classe Digitaleo est hébergé sur github:
// https://github.com/digitaleo/Digitaleo-api-php

$digitaleoApi = new Digitaleo();

$digitaleoApi->setClientCredentialsGrant(
   '<your client_id>',
   '<your client_secret>');

/**
* Importer le fichier audio sur la plateforme "Digitaleo"
**/

$digitaleoApi->setBaseUrl('https://baseo.messengeo.net/content/rest/');
$uploadResult = $digitaleoApi->contentsCreate(
   array(
       'name'        => 'Soldes2014',
       'description' => 'Campagne vocale pour les soldes hiver 2014',
       'tags'        => array('soldes','promotions')
   ),
   array(
       'contentFile' => '/users/robert/soldes.wav'
   )
);

$audioUrl = $uploadResult->list[0]->url_encoded;

/**
* Créer le mailing vocal
**/

$digitaleoApi->setBaseUrl('https://api.messengeo.net/rest/');
$sendResult = $digitaleoApi->mailingsCreate(
   array(
       'name'  => 'Campagne Vocale Soldes 2014',
       'link'  => $audioUrl,
       'media' => 'VOICEMAIL',
       'contacts' => array(
           array('recipient' => '+33612345678'),
           array('recipient' => '+33612345679'),
           array('recipient' => '+33612345680')
       )
   )
);
```

4. Retrieving the deliverability status of messages

When a ping URL is specified (when the mailing is created), the Digitaleo platform calls this URL by replacing the pattern #id# (or #ids#) with the message ids for which the status has changed.

Using the id specified, it is then possible to retrieve the status of the message via the "messages" resource

4.1. List of parameters to supply in order to read a message

Property	**Description**
id *required*	Allows you to filter according to one or more message ids
properties *required*	By default, the message resource contains about fifteen attributes that is not necessary to read when you only want to retrieve the current status of the message. This parameters allows you to reduce the list of attributes of the resource which improves the response time. In our case, the attributes of interest are ● the identifier - id ● the current status - currentStatus

4.2. List of statuses

status	Description
wait	Message waiting to be sent - Message programmed
on	Message currently being sent
ok	Message delivered
ko	Message not delivered
no	Message for which the deliverability status is not known

4.3. Example of processing the ping URL with PHP

```php
<?php

// Le code source de la classe Digitaleo est hébergé sur github:
// https://github.com/digitaleo/Digitaleo-api-php

$voiceApi = new Digitaleo('https://api.messengeo.net/rest/');

$voiceApi->setClientCredentialsGrant(
   '<your client_id>',
   '<your client_secret>'
);

$readResult = $voiceApi-->messagesRead(
   array(
       'id'         => explode('-',$_GET['ids']),
       'properties' => array('id','currentStatus'),
   )
);

foreach ($readResult->list as $message) {
   printf('%s -> %s (%s)',
       $message->id,
       $message->currentStatus->shortKey,
       $message->currentStatus->description);
}
```

The below would produce the following output:

```
123456 --> ko (voice busy)
789123 --> ok (voicemail delivered)
```

5. Retrieving deliverability statistics

5.1. Lists of various statistics

The API allows you to retrieve the sending statistics for a mailing and more particularly the following elements:

status	**Description**
total	Number of messages of the mailings
wait	Number of messages waiting to be sent
on	Number of messages currently being sent
ok	Number of messages delivered
ko	Number of messages not delivered
no	Number of messages for which the deliverability status is not known
optout	Number of messages from which the recipients have unsubscribed from the service *only for the VOICE media*

These elements are present in the *stats* attribute of the *mailing* resource.

5.2. Example 1. Retrieving the statistics of a particular mailing

```
GET /rest/mailings?properties=id,stats&id=71538 HTTP/1.1
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpZCI6ImYyMzE2…
Host: api.messengeo.net
```

with Curl

```
curl
-X GET
-H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbG...ccOqbVow8xOQyQ"
-d properties=id,stats
-d id=71538
https://api.messengeo.net/rest/mailings
```

return

```
{
  "size": 1,
  "total": 1,
  "list": [
  {
    "id": "71538",
    "stats": {
      "total": 98973,
      "wait": 0,
      "on": 0,
      "ok": 90224,
      "ko": 8687,
      "no": 62,
      "optout": 0,
      "rep": 0
    }
  }
  ],
  "httpStatusCode": 200
}
```

5.3. Example 2. Statistics for all mailings sent with a particular name

```php
<?php

$clientId     = 'b895af26fc3c4436efb7ebcf197e9923';
$clientSecret = 'd5d7f2e2a4dd516cfcbadaf650f52e2d';
$mailingName  = '<Nom de mon mailing>';

$digitaleoApi = new Digitaleo('https://api.messengeo.net/rest');
$digitaleoApi->setClientCredentialsGrant($clientId, $clientSecret);

$res = $digitaleoApi->mailingsRead(
   array(
       'name'       => $mailingName,
       'properties' => 'id,stats'
   )
);

$total = $ok = 0;

foreach ($res->list as $mailing) {
   $ok    += $mailing->stats->ok;
   $total += $mailing->stats->total;
}

if ($total > 0) {
   printf('Délivrabilité du mailing "%s": %0.2f %%' . "\n", $mailingName, $ok * 100 /
$total);
}
```

Copyright