

**SAPO Services SDK Documentation**

**v0.2**

Contents

[1 Introduction 4](#_Toc336616993)

[2 Authentication 4](#_Toc336616994)

[2.1 HTTP Clients 4](#_Toc336616995)

[2.1.1 Using username and password 4](#_Toc336616996)

[2.1.2 Using authentication token 4](#_Toc336616997)

[3 Photos Service 4](#_Toc336616998)

[3.1 HTTP 5](#_Toc336616999)

[3.1.1 Submit a photo to the service 5](#_Toc336617000)

[3.1.2 Search images by terms 17](#_Toc336617001)

[3.1.3 Delete a photo 17](#_Toc336617002)

[3.1.4 Get photo details 17](#_Toc336617003)

[3.1.5 Get user’s photos 17](#_Toc336617004)

[3.1.6 Create an Album 17](#_Toc336617005)

[3.1.7 Add an photo to albums 17](#_Toc336617006)

[3.1.8 Get album images 17](#_Toc336617007)

[4 Videos Service 17](#_Toc336617008)

[4.1 HTTP 17](#_Toc336617009)

[4.1.1 Submit a video to the service 17](#_Toc336617010)

[4.1.2 Delete a video 17](#_Toc336617011)

[4.1.3 Get video details 17](#_Toc336617012)

[4.1.4 Get user details 17](#_Toc336617013)

[4.1.5 Get user videos 17](#_Toc336617014)

[5 Verbetes Service 17](#_Toc336617015)

[5.1 HTTP 17](#_Toc336617016)

[5.1.1 Who is 17](#_Toc336617017)

[5.1.2 Get personalities 17](#_Toc336617018)

[5.1.3 Get jobs 17](#_Toc336617019)

[5.1.4 Get egocentric network of personality 17](#_Toc336617020)

[5.1.5 Get personalities co-occurrences 17](#_Toc336617021)

[5.1.6 Get number of times that two personalities co-occur 17](#_Toc336617022)

[6 Auto Service 17](#_Toc336617023)

[6.1 HTTP 17](#_Toc336617024)

[6.1.1 Search by terms 17](#_Toc336617025)

[6.1.2 Search by brand, model, price 17](#_Toc336617026)

[7 PunyURL Service 17](#_Toc336617027)

[7.1 HTTP 17](#_Toc336617028)

[7.1.1 Get compressed URI 17](#_Toc336617029)

[7.1.2 Get URI from compressed URI 17](#_Toc336617030)

[8 Captcha Service 17](#_Toc336617031)

[8.1 HTTP 17](#_Toc336617032)

[8.1.1 Get captcha image 17](#_Toc336617033)

[8.1.2 Get captcha audio 17](#_Toc336617034)

# Introduction

The present document describes the SAPO Services Windows 8 SDK.

Before you can use a service you must create an account at <https://store.services.sapo.pt/>. After creating the account you can acquire services subscriptions. In the customer area you can manage you services and generate authorization information, i.e. an access key.

# Authentication

Every request must contain authentication information to establish the identity of whom making the request and an authorization token denominated **ESBAccessKey**.

The authentication information can be provided as a pair of username (**ESBUsername**) and password (**ESBPassword**) or an authentication token (**ESBToken**) obtained from the **SAPO Security Token Service** (STS) <https://store.services.sapo.pt/en/Catalog/development/free-api-sts>.

If you don't want to constantly send passwords "over the wire" you should use STS. This way you only send the credentials once.

## HTTP Clients

As previously said, every request must include both authentication and authorization information. The authentication information must be supplied in the URI. The authorization information must figure in the **Authorization** HTTP Header.

Note that in the samples below {token}, {username}, {pass}, and {accessKey} are place holders. In your code you must replace them with your authorization and authentication data.

### Using username and password

The following example presents the structure of a request using **ESBUsername** and **ESBPassword** to do the authentication. Authorization and Host are HTTP headers.

GET https://services.sapo.pt/{service\_name}/{operation\_name}?ESBUsername={username}&ESBPassword={pass}&json=true HTTP/1.1

Authorization: ESB AccessKey={accessKey}

Host: services.sapo.pt

### Using authentication token

The following example presents the structure of a request using **ESBToken** to do the authentication. Authorization and Host are HTTP headers.

GET https://services.sapo.pt/{service\_name}/{operation\_name}?ESBToken={token}&json=true HTTP/1.1

Authorization: ESB AccessKey={accessKey}

Host: services.sapo.pt

# Photos Service

The **Photos** service allows to search and to manage of photos that are hosted in **SAPO Photos** (<http://fotos.sapo.pt/>).

To interact with the service are available a SOAP interface and a HTTP-JSON interface.

## HTTP

This section describes the HTTP-JSON interface of the **Photos** service

In all the samples of this chapter it is used **ESBUsername** and **ESBPassword** to do the authentication. But as mentioned in the previous chapter, alternatively you can provide **ESBToken**.

### Submit a photo to the service

To submit a photo to the service you can use the **ImageCreate** operation.

The **ImageCreate** operation is composed by two steps. First you have to send the meta data associated with the photo through a HTTP POST to <https://services.sapo.pt/Photos/ImageCreate>. Second you have to do a HTTP POST to <http://fotos.sapo.pt/uploadPost.html> with the photo file.

#### POST the meta data of the photo

The following example is a sample request of the POST of the meta data of the photo.

POST https://services.sapo.pt/Photos/ImageCreate?json=true&ESBUsername={username}&ESBPassword={password} HTTP/1.1

Content-Type: application/json

Authorization: ESB AccessKey={accessKey}

Content-Length: 48

Host: services.sapo.pt

{"image":{"title":"windows8","tags":"windows8"}}

For the sake of simplicity there are only provided the title and the tags of the image. It's recommended that when you submit a new photo you provide at least this attributes.

The complete list of Image attributes can be found at [https://store.services.sapo.pt/en/Catalog/social/free-api-photos/technical-description#entity-type-Photos-Image](https://store.services.sapo.pt/en/Catalog/social/free-api-photos/technical-description%23entity-type-Photos-Image). But if you go forward a couple of pages you will find a example with all the Image object attributes.

The service response will have a similar structure to the one in example below. In the body of the response figures an object that has a **ImageCreateResponse** attribute. The value attribute is an **ImageCreateResult** object that as two attributes: image and result.

HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

Date: Tue, 18 Sep 2012 10:14:54 GMT

Content-Length: 1014

{

"ImageCreateResponse":{

"ImageCreateResult":{

"image":{

"active":"true",

"creationDate":"2012-09-18 11:00:43",

"innapropriate":"false",

"m18":"false",

"password":null,

"pending":"true",

"rating":"0",

"subtitle":null,

"synopse":null,

"tags":"sapo",

"title":"sapo",

"uid":"{...}",

"url":"http://fotos.sapo.pt/{...}/fotos/?uid={...}",

"user":{

"avatar":"http://imgs.sapo.pt/sapofotos/{...}/imgs/avatar.jpg",

"url":"http://fotos.sapo.pt/{...}/perfil",

"username":"{...}"

},

"visualizations":"0"

},

"result":{

"ok":"true"

},

"token":"{photo\_token}"

}

}

}

The image is an instance of **Image** type, which has the meta data supplied and another server generated fields, like **uid** that identifies the photo.

You can see if the request was accepted by the server looking to the **ok** attribute of the **result** object. The **{photo\_token}** attribute is the token that you will have to send along with the photo file.

Note that **{photo\_token}** is place holder and **{...}** is censuring sensitive data like usernames and photo ids.

#### POST the photo file

The second and final step of the upload of the photo is a **multipart** POST.

The structure of the request can be seen in following example.

POST http://fotos.sapo.pt/uploadPost.html HTTP/1.1

Content-Type: multipart/form-data; boundary="imgboundary"

Host: fotos.sapo.pt

Content-Length: 75195

--imgboundary

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

{photo\_token}

--imgboundary

Content-Type: image/png

Content-Disposition: form-data; name="photo"; filename="windows8.png"

{photo\_bytes}

Note that **{photo\_token}** and **{photo\_bytes}** are place holders. The first one is the token returned by the server in the **ImageCreateResult** object. The other is the bytes of the photo file.

If all goes as it should in the response you will get a response with **XML** content where you can find a **Result** tag with "SUCCESS" in the content. Otherwise you will get a response with a smaller **XML** document that has in the **Result** tag the error code. You can see the complete error list at [https://store.services.sapo.pt/en/Catalog/social/free-api-photos/technical-description#service-Photos-operation-ImageCreate](https://store.services.sapo.pt/en/Catalog/social/free-api-photos/technical-description%23service-Photos-operation-ImageCreate).

In the following example you can see a sample response body. In this case the photo was successfully submitted. So in the **XML** document will be present all the **views** generated. Each **view** has a URI (**url** tag) to the photo file. Along with the URI is also provided the **with** and the **height** of the photo.

<?xml version="1.0"?>

<uploadPost>

<Ok/>

<Result>SUCCESS</Result>

<views>

<view>

<size>large</size>

<requestWidth>1600</requestWidth>

<requestHeight>1200</requestHeight>

<url>http://{...}.png</url>

</view>

(...)

</views>

</uploadPost>

In case of error, for instance if you provide an invalid token the body of the response will look like the one in following example.

<?xml version="1.0" encoding="utf-8" ?>

<UploadPost>

<Result>INVALID\_TOKEN</Result>

<Error>Invalid token</Error>

</UploadPost>

**Important:** Note that in case of error the **UploadPost** tag begin with capital letter.

### Search images by terms

To search images by terms you can use the **ImageGetListBySearch** operation.

To do this search you have to do a HTTP GET to the following URI: <http://services.sapo.pt/Photos/ImageGetListBySearch?json=true>.

The search terms must be specified in the **string** query string parameter. But it’s recommended that you can provide additional search information like dates. The complete list of query string parameters that you can provide to this service are:

* **string** – the search terms separated by commas and with no spaces (e.g. “windows,microsoft”);
* **datefrom** – the date from the search results begin in the format “yyyy-mm-dd” (e.g. 2012-10-15);
* **dateto** – the date to the search results end in the format “yyyy-mm-dd” (e.g. 2012-10-17);
* **page** – the results supports paging, in this parameter you can specify the page. (e.g. “1”).

**Important:** Note that it’s mandatory to include **json** parameter and it must be set with **true**. And like all other requests the authentication date must be provided in the query string.

The following example is presented a sample request:

GET http://services.sapo.pt/Photos/ImageGetListBySearch?string=windows,microsoft&page=1&datefrom=2012-09-10&dateto=2012-09-12&json=true&ESBUsername={username}&ESBPassword={pass} HTTP/1.1

Authorization: ESB AccessKey={accessKey}

Host: services.sapo.pt

The service response body will have a similar structure to the one in example below. In the body of the response figures an object that has an **ImageGetListBySearchResponse** attribute. The value attribute is a **ImageGetListBySearchResult** object that as two attributes: **images** and **result**. The **images** attribute is a array of **image** objects.

{

"ImageGetListBySearchResponse":{

"ImageGetListBySearchResult":{

"images":{

"image":[

{

"active":"true",

"creationDate":"2011-09-14 22:56:00",

"innapropriate":"false",

"m18":"false",

"password":null,

"pending":"false",

"subtitle":null,

"synopse":"Windows 8",

"tags":"windows 8",

"title":"windows\_8",

"uid":"{...}",

"url":"http://fotos.sapo.pt/{...}/fotos/?uid={...}",

"user":{ "avatar":"http://imgs.sapo.pt/sapofotos/{...}/imgs/avatar.jpg",

"url":"http://fotos.sapo.pt/{...}/perfil",

"username":"{...}"

},

"views":{

"view":[

{

"requestHeight":"405",

"requestWidth":"540",

"size":"original",

"url":"http://{...}.jpeg"

},

(...)

]

}

},

(...)

]

},

"result":{

"ok":"true",

"page":"1",

"perPage":"50",

"total":"41",

"totalPages":"1"

}

}

}

}

If you pretend to download the photo, in the **image** object’s **views** attribute is provided and array of **view** objects. Each view is a version of the photo with different dimensions (with and height). The **url** attribute has the URI from you can GET the photo file.

### Delete a photo

To delete a photo you can use the **ImageDelete** operation.

To do a delete of a photo you have to do a HTTP GET to the following URI: <https://services.sapo.pt/Photos/ImageDelete?json=true>.

The **uid** of the photo must be included in the query string parameters.

The following example is a sample request to delete the photo with the **uid** {id}.

GET http://services.sapo.pt/Photos/ImageDelete?uid={id}json=true&ESBUsername={username}&ESBPassword={pass} HTTP/1.1

Authorization: ESB AccessKey={accessKey}

Host: services.sapo.pt

If the photo is successfully deleted the server in the response body will send an json object similar to the one below:

{

"ImageDeleteResponse": {

"ImageDeleteResult":{"ok":"true"}

}

}

### Get photo details

To get the details of a photo you can use the **ImageDetails** operation.

In order to get the details of the photo you have to do a HTTP GET to the following URI: <https://services.sapo.pt/Photos/ImageDetails?json=true>.

The **uid** of the photo must be included in the query string parameters.

The following example is a sample request to get the details of the photo with the **uid** {id}.

GET http://services.sapo.pt/Photos/ImageDetails?uid={id}json=true&ESBUsername={username}&ESBPassword={pass} HTTP/1.1

Authorization: ESB AccessKey={accessKey}

Host: services.sapo.pt

The response body will be similar to the one in the following example:

{

"ImageDetailsResponse":{

"ImageDetailsResult":{

"image":{

"active":"true",

"creationDate":"2012-09-05 11:16:03",

"innapropriate":"false",

"m18":"false",

"password":null,

"pending":"false",

"subtitle":null,

"synopse":null,

"tags":"stack,windows",

"title":"Windows 8 stack",

"uid":"{...}",

"url":"http://fotos.sapo.pt/{...}/fotos/?uid={...}",

"user":{

"avatar":"http://imgs.sapo.pt/sapofotos/{...}/imgs/avatar.jpg",

"url":"http://fotos.sapo.pt/{...}/perfil",

"username":"test8"

},

"views":{

"view":[

{

"requestHeight":"581",

"requestWidth":"1024",

"size":"original",

"url":"http://{...}.jpeg"

},

(....)

]

}

},

"result":{

"ok":"true"

}

}

}

}

### Get user’s photos

To get the user’s photos you can use the **ImageGetListByUser** operation.

In order to get the list of photos of the user you have to do a HTTP GET to the following URI: <https://services.sapo.pt/Photos/ImageGetListByUser?json=true>.

The **username** of the user should be included in the query string parameters. If you don’t specify the username the service will assume that the user is the authenticated user.

The following example is a sample request to get the photos list of the user with the **username** {username}.

GET http://services.sapo.pt/Photos/ImageGetListByUser?username={username}json=true&ESBUsername={username}&ESBPassword={pass} HTTP/1.1

Authorization: ESB AccessKey={accessKey}

Host: services.sapo.pt

The response body will be similar to the one in the following example:

{

"ImageGetListByUserResponse":{

"ImageGetListByUserResult":{

"images":{

"image":[

{

"active":"true",

"creationDate":"2012-09-18 11:00:42",

"innapropriate":"false",

"m18":"false",

"password":null,

"pending":"false",

"subtitle":null,

"synopse":null,

"tags":null,

"title":"sapo",

"uid":"{...}",

"url":"http://fotos.sapo.pt/{...}/fotos/?uid={...}",

"views":{

"view":[

{

"requestHeight":"450",

"requestWidth":"450",

"size":"original",

"url":"http://{...}.png"

},

(...)

]

}

},

(...)

]

},

"result":{

"ok":"true",

"page":"1",

"perPage":"50",

"total":"11",

"totalPages":"1"

}

}

}

}

Note that if the photos are in an album the **image** object will have an attribute **albums** with the albums that the photo belongs to.

### Create an Album

To create an album you can use the **AlbumCreate** operation.

The creation is done doing a HTTP POST to the following URI: <https://services.sapo.pt/Photos/AlbumCreate?json=true>. You must provide a JSON object with the album representation.

The following example is a sample request to create an album:

POST https://services.sapo.pt/Photos/AlbumCreate?json=true&ESBUsername={username}&ESBPassword={pass} HTTP/1.1

Host: services.sapo.pt

Authorization: ESB AccessKey={accessKey}

Content-Length: 30

Content-Type: application/json

{"album":{"title":"Windows8"}}

The complete list of the **album** attributes can be found at <https://store.services.sapo.pt/en/Catalog/social/free-api-photos/technical-description#entity-type-Photos-Album>.

If the album is successfully created the response body of the request will be similar to the one in the following example:

{

"AlbumCreateResponse":{

"AlbumCreateResult":{

"album":{

"active":"true",

"cover":"http://fotos.sapo.pt/{...}/capa-de-album/4",

"creationDate":"2012-09-19 12:51:03",

"description":null,

"id":"4",

"ownerId":"181291",

"password":null,

"title":"Windows8",

"url":"http://fotos.sapo.pt/{...}/albuns/?aid=4",

"user":{

"avatar":"http://imgs.sapo.pt/sapofotos/{...}/imgs/avatar.jpg",

"url":"http://fotos.sapo.pt/{...}/perfil",

"username":"{...}"

}

},

"result":{

"ok":"true"

}

}

}

}

### Add an photo to albums

To add a photo to albums you can use the **AlbumCreate** operation.

To do this you have to do a GET request to the following URI: <https://services.sapo.pt/Photos/ImageAddToAlbum?json=true>.

In the query string you have to specify the **image** **uid** and the **albums** **ids**. This is done with the following query string parameters:

* **imageuid** – **image uid**;
* **albumid** – **albums ids** separated by commas (e.g. “2,3”);

The following example is a sample request that adds the photo with the **uid {id}** to **albums** **2** and **3**.

GET https://services.sapo.pt/Photos/ImageAddToAlbum?ESBUsername={user}&ESBPassword={pass}&imageuid={id}&albumid=2,3&json=true HTTP/1.1

Host: services.sapo.pt

Authorization: ESB AccessKey={accessKey}

If the photos are successfully added to the albums, the response body will be similar to the one in the following example:

{

"ImageAddToAlbumResponse":{

"ImageAddToAlbumResult":{

"result":{

"ok":"true"

}

}

}

}

### Get album images

To get a user’s album photos list you can use the **ImageGetListByUserAlbum**.

You have to do a HTTP GET to the following URI specifying the **SAPO Photos username** and the **album id**. The First should be passed as username query string parameter and the last as the id query string parameter.

The following example is a sample request to get the photos list of the **album** with **id 2** of the **user** with **SAPO Photos username** **{sapofotos\_username}**:

GET https://services.sapo.pt/Photos/ImageGetListByUserAlbum?ESBUsername={username}&ESBPassword={pass}&username={sapofotos\_username}&id=2&page=1&json=true HTTP/1.1

Host: services.sapo.pt

Authorization: ESB AccessKey={accessKey}

Content-Type: application/json

The response body is similar to the one in the following example:

{

"ImageGetListByUserAlbumResponse":{

"ImageGetListByUserAlbumResult":{

"images":{

"image":[

{

"active":"true",

"creationDate":"2012-09-04 12:58:45",

"innapropriate":"false",

"m18":"false",

"password":null,

"pending":"false",

"subtitle":null,

"synopse":null,

"tags":null,

"title":"microsoft",

"uid":"{...}",

"url":"http://fotos.sapo.pt/{...}/fotos/?uid={...}",

"views":{

"view":[

{

"requestHeight":"465",

"requestWidth":"620",

"size":"original",

"url":"http://{...}.jpeg"

},

(...)

]

}

},

(...)

]

},

"result":{

"ok":"true",

"page":"1",

"perPage":"50",

"total":"2",

"totalPages":"1"

}

}

}

}

# Videos Service

The **Videos** service allows to search and to manage videos that are hosted in **SAPO Videos** (<http://videos.sapo.pt/>);

To interact with the service is available a HTTP interface and a SOAP interface.

## HTTP

This section describes the HTTP-JSON interface of the **Videos** service.

### Submit a video to the service

### Delete a video

### Get video details

### Get user details

### Get user videos

# Verbetes Service

**Verbetes** is a service that answers to requests of type "WhoIs" for public personalities. All the information from **Verbetes** is collected automatically from news sources, and its information is updated on a hour basis, as new news are collected. **Verbetes** also answers to requests of type "WhoIs" for jobs of public personalities mentioned on news.

To interact with the service is available a HTTP-JSON interface.

## HTTP

This section describes the HTTP-JSON interface of the **Verbetes** service.

In all the samples of this chapter it is used **ESBUsername** and **ESBPassword** to do the authentication. But as previously mentioned, alternatively you can provide **ESBToken**.

### Who is

To find “verbetes” about public personalities or jobs you can use the **WhoIs** operation.

To do this search

GET http://services.sapo.pt/InformationRetrieval/Verbetes/WhoIs?name={name}&name\_like={name\_like}&job={job}&job\_like={job\_like}&date=2012-11-15&margin=5&min=5&format=json&ESBUsername={username}&ESBPassword={pass} HTTP/1.1

Authorization: ESB AccessKey={accessKey}

Host: services.sapo.pt

Note that exists a light version of the **WhoIs** operation that’s designed to be faster. It is similiar to the **WhoIs**, so the considerations of this subsection also apply to the light version (**WhoIsLight**).

### Get personalities

To get all personalities that have at least **n** “verbetes” you can use the **GetPersonalities** operation.

### Get jobs

To get all jobs that have at least **n** occurrences you can use the **GetErgos** operation.

### Get egocentric network of personality

To get the egocentric network of some personality you can use the **GetEgoNet** operation.

### Get personalities co-occurrences

To get the personalities who co-occur with a given personalitie you can use the **GetCoOccurences** operation.

### Get number of times that two personalities co-occur

To get the number of co-occurences of two given personalities you can use the **GetCoOccurrencesTrends** operation.

# Auto Service

The **Auto** service allows search of auto vehicles in the contents of **SAPO Auto** (<http://auto.sapo.pt/>).

To interact with the service is available a HTTP interface with both JSON and XML formats.

## HTTP

This section describes the HTTP-JSON interface of the **Auto** service.

### Search by terms

To search for auto vehicles by given terms you can use **JSON** operation freely specifying the terms in the **q** query string parameter.

### Search by brand, model, price

To search for auto vehicles by brand, model or price you can use the **JSON** operation.

# PunyURL Service

The **PunyURL** service allows to, given an URI, obtain two compressed URI's: One in Unicode basead on RFC3492 of PunyCode and other with alphanumeric characters in lowercase.

To interact with the service is available a HTTP interface with both JSON and XML formats.

## HTTP

This section describes the HTTP-JSON interface of the **PunyURL** service.

### Get compressed URI

To get a compressed version of a given URI you can use the **GetCompressedURLByURLJSON** operation.

### Get URI from compressed URI

To get the URI that corresponds to a given compressed URI you can use the **GetURLByCompressedURL** operation.

# Captcha Service

The **Captcha** service allows the generation and showing of a customizable captcha.

To interact with the service is available a HTTP-JSON interface.

## HTTP

This section describes the HTTP-JSON interface of the **Captcha** service.

You can get captchas in audio or image formats. And note that for the same captcha you can present it as a image and as a audio.

### Get captcha image

The process of obtaining a image captcha requires two diferent steps. First you have to generate the captcha using the **Get** operation. Then you can obtain the captcha image using the **Show** operation and specifying the **id** of the generated captcha.

### Get captcha audio

The process of obtaining a audio captcha requires two diferent steps. First you have to generate the captcha using the **Get** operation. Then you can obtain the captcha audio using the **Play** operation and specifying the **id** of the generated captcha.

# 