



# Technical Note REST API Description

---

REST API description. Version 1.1. Jul 2017.



## Document's information

---

<b>Project</b>	<b>BITEXT POS TAGGING API</b>
<b>Description</b>	REST API description. Version 1.1. Jul 2017.
<b>Author (s)</b>	Luis Miguel Serrano; Luis Caloto
<b>Customer / Dept.</b>	
<b>Last Mod. Date</b>	July 12 <sup>th</sup> , 2017
<b>Issue</b>	A
<b>DOC_ID</b>	

---

## Table of contents

<b>1</b>	<b>Purpose</b> .....	<b>4</b>
<b>2</b>	<b>Scope</b> .....	<b>4</b>
<b>3</b>	<b>Documents</b> .....	<b>4</b>
3.1	Applicable Documents.....	4
3.2	Reference Documents.....	4
<b>4</b>	<b>Definitions and Acronyms</b> .....	<b>5</b>
4.1	Definitions.....	5
4.2	Acronyms and Abbreviations.....	5
<b>6</b>	<b>REST API</b> .....	<b>¡Error! Marcador no definido.</b>
<b>7</b>	<b>POS Tagging Analysis</b> .....	<b>9</b>
7.1	POST Request .....	9
7.2	GET Request.....	9
7.3	Example.....	11
<b>8</b>	<b>Code Samples</b> .....	<b>13</b>
8.1	Perl .....	13
8.2	Python .....	15
8.3	Ruby .....	17
8.4	Java .....	19

## 1 Purpose

This document describes how to use the Bitext POS Tagging API.

## 2 Scope

This document applies to the integration and adaptation of the Bitext POS Tagging API, provided by Bitext.

Descriptions and instructions contained in this document might be modified in future versions of this same document.

## 3 Documents

### 3.1 Applicable Documents

---

Code	Name	Version
N/A		

---

### 3.2 Reference Documents

---

Code	Name	Version
N/A		

---

## 4 Definitions and Acronyms

### 4.1 Definitions

---

### 4.2 Acronyms and Abbreviations

<b>ASCII</b>	<i>American Standard Code for Information Interchange</i> (character encoding set)
<b>API</b>	<i>Application Programming Interface</i>
<b>SW</b>	<i>Software</i>
<b>CSV</b>	<i>Comma Separated Values</i> – (structured plain text format)
<b>UTF-8</b>	<i>UCS Transformation Format – 8 bits</i> (character encoding set)
<b>UCS</b>	<i>Universal Character Set</i> (character set standard)
<b>ISO-8859-1</b>	Character encoding set based on Latin character set for most Western European languages used mainly in Unix/Linux Operating Systems
<b>Windows1252</b>	Character encoding set based on Latin character set for most Western European languages used mainly in Windows Operating Systems

## 6 REST API

The Bitext POS Tagging API can be reached by using the following endpoint:

<https://svc02.api.bitext.com/postagging/>

For this API, the following languages are available: Standard Arabic (MSA), Catalan, Croatian, Czech, Danish, Dutch, English, French, German, Hungarian, Italian, Korean, Norwegian Bokmål, Norwegian Nynorsk, Polish, Portuguese, Romanian, Russian, Persian (Farsi), Serbian, Slovak, Slovenian, Spanish, Swedish, Turkish and Ukrainian.

You must specify the language code as parameter for requests:

Language	Code
Standard Arabic (MSA)	arb
Catalan	cat
Croatian	hrv
Czech	ces
Danish	dan
Dutch	nld
English	eng
French	fra
German	deu
Hungarian	hun
Italian	ita
Korean	kor
Norwegian Nynorsk	nno
Norwegian Bokmål	nob
Persian (Farsi)	fas
Polish	pol
Portuguese	por
Romanian	ron
Russian	rus
Serbian	srp
Slovak	slk
Slovenian	slv

<b>Spanish</b>	spa
<b>Swedish</b>	swe
<b>Turkish</b>	tur
<b>Ukrainian</b>	ukr

A person identification token should be added to all requests. Such a token is typically provided by Bitext after creating an account in the API website, below My Profile > API Credentials. For this document we'll use the example token "tokenprovidedbybitext".

The token must be sent as the value of the Authorization header, and should appear with a bearer. For example, using "curl", you can reach the Bitext API like this:

```
curl -X POST --url https://svc02.api.bitext.com/postagging/ -H "Authorization: bearer tokenprovidedbybitext"
```

If you expect a JSON response, you must also specify so in the headers. You must instantiate the header "Content-Type" with the "application/json" value.

```
curl -X POST --url https://svc02.api.bitext.com/postagging/ -H "Authorization: bearer tokenprovidedbybitext" -H "Content-Type: application/json"
```

The Bitext API works asynchronously. You must first request the API to start analyzing a text. After this first request, you can perform additional requests, until the API eventually returns the correct analysis for the text included in the first request.

The first request is a POST request that sends a JSON with two parameters: language and text. The value of the text parameter corresponds to the text to be analyzed, and the value of the language parameter corresponds to the language of this text.

Currently, the limit of characters per API call is 50.000

For example, in "curl", a first request for the analysis of "Your house is great" will look as follows:

- ```
curl -X POST --url https://svc02.api.bitext.com/postagging/ -H "Authorization: bearer tokenprovidedbybitext" -H "Content-Type: application/json" --data '{"language":"eng", "text":"Your house is great"}'
```

This request is calling to the "postagginganalysis" endpoint. The API will immediately return a JSON response that looks like this:

```
{
```

```
"resultid": "9a23759db4ae46d8bfbbab47ac4dff1f",
"success": true,
"message": "Request accepted"
}
```

The **resultid** field in this response is crucial. It will be used to build the subsequent requests asking if the result for a given previous request is already available or not. These new requests are GET type (not POST) and they **MUST** include the **resultid** in the URL. To ask the Bitext API if the results for one of the texts previously sent, you would write the following request with “curl”:

```
curl -X GET --url https://svc02.api.bitext.com/postagging/9a23759db4ae46d8bfbbab47ac4dff1f/ -H "Authorization: bearer tokenprovidedbybitext" -H "Content-Type: application/json"
```

If the analysis is already available, the response to that request would be the following:

```
{
  "resultid": "9a23759db4ae46d8bfbbab47ac4dff1f",
  "postagginganalysis": [
    [
      {
        "form": "Your",
        "pos": "determiner"
      },
      {
        "form": "house",
        "pos": "noun"
      },
      {
        "form": "is",
        "pos": "verb"
      },
      {
        "form": "great",
        "pos": "adjective"
      }
    ]
  ]
}
```

If the analysis is not yet available the (202 HTTP code) will be returned, and you will need to try the same request again.



## 7 POS Tagging Analysis

The Bitext POS Tagging service provides the Part of Speech (POS) for each one of the words in a sentence. When a word can have several POSes, the relevant POS will be decided using syntactic and morphological analysis. For example, for the sentence "I want to run", the POS Tagging service will return the "pronoun" for "I", "verb" for "want" and "run", and "preposition" for "to". But in "It was a good run", "run" will be assigned the POS "noun".

### 7.1 POST Request

#### Request

- Endpoint: /postagging/
- Protocol: HTTPS
- URL: <https://svc02.api.bitext.com/postagging/>
- Method: POST
- Headers:

```
{
  "Authorization": "bearer tokenprovidedbybitext",
  "Content-Type": "application/json"
}
```

- Data parameters:

```
{
  "language": "eng",
  "text": "..."
}
```

Being "**text**", the string you want to analyze.

Successful responses:

- Code: 201

```
{
  "success": true,
  "message": "Request accepted",
  "resultid": "..."
}
```

### 7.2 GET Request

#### Request

- Endpoint: /postagging/
- Protocol: HTTPS
- URL: https://svc02.api.bitext.com/postagging/:resultid/
- Method: GET
- Headers:

```
{
  "Authorization": "bearer tokenprovidedbybitext",
  "Content-Type": "application/json"
}
```

Being “**resultid**” the identifier of the analysis request (retrieved by the POST transaction)

Successful responses:

- Code: 202

The analysis with ID “**resultid**” is not yet complete and must be requested again.

```
{
  "resultid": "...
}
```

- Code: 200

```
{
  "success": true,
  "resultid": "...",
  "postagginganalysis": [
    [
      {
        "form": "...",
        "pos": "...
      },
      {
        "form": "...",
        "pos": "...
      },
      {
        "form": "...",
        "pos": "...
      }
    ]
  ]
}
```

- Code 200: The analysis with ID “**resultid**” is complete

- resultid: Identifier of the analysis request (retrieved by the POST transaction and sent in the current request)
- form: each of the words in the text
- pos: the Part of Speech of the word in that sentence. POSes can be "noun", "adjective", "verb", "adverb", "determiner", "pronoun", "preposition" or "conjunction", among others.

## 7.3 Example

### POST Request

```
-s -X POST --url https://svc02.api.bitext.com/postagging/ -H "Authorization: bearer tokenprovidedbybitext" -H "Content-Type: application/json" -- data '{"language":"eng", "text":"Your house is great"}'
```

### POST Response

```
{  
  "success": true,  
  "message": "Request accepted",  
  "resultid": "123451234512345123451234512345ab"  
}
```

### GET Request

```
curl -s -X GET --url https://svc02.api.bitext.com/postagging/123456789abcdefg123456789abcdefg/ -H "Authorization: bearer tokenprovidedbybitext" -H "Content-Type: application/json"
```

### GET Response

```
{  
  "resultid": "1b84e78a8655448db6b1730927fba1a2",  
  "postagginganalysis": [  
    [  
      {  
        "form": "your",  
        "pos": "determiner"  
      },  
      {  
        "form": "house",  
        "pos": "noun"  
      },  
      {  
        "form": "is",  
        "pos": "verb"  
      }  
    ]  
  ]  
}
```

```
{
  "form": "great",
  "pos": "adjective"
}
]
```

## 8 Code Samples

### 8.1 Perl

```
use Mojo::UserAgent;
use Mojo::JSON qw(encode_json);

# User token, required for API access
my $oauth_token = "tokenprovidedbybitext";

# API request data: Language and text to be analyzed
my $user_language = "eng";
my $user_text = "Your house is great";

print "\nBITEXT API. POS Tagging endpoint. PERL sample code\n";
print "-----\n\n";

my $ua = Mojo::UserAgent->new;

# Building the POST request to POS Tagging analysis endpoint
my $endpoint = "https://svc02.api.bitext.com/posttagging/";
my $headers = { Authorization => "bearer $oauth_token", 'Content-Type' => 'application/json' };
my $params = {"language" => "$user_language", "text" => "$user_text"};

# Sending the POST request
$tx = $ua->post($endpoint, $headers, json => $params);

# Processing the result of the POST request
my $post_result = $tx->res->json->{success};           # Success of the request
my $post_result_code = $tx->res->code;                 # Error code, if applicable
my $post_msg = $tx->res->json->{message};              # Error message, if applicable
my $action_id = $tx->res->json->{resultid};            # Identifier to request the analysis results

print "POST: '$post_msg'\n\n";

# 401 is the error code corresponding to an invalid token
if ($post_result_code == 401)
{
    print "Your authentication token is not correct\n";
}

if ($post_result)
{
    print "Waiting for analysis results...\n\n";
}
```

```
answer # GET request loop, using the response identifier returned in the POST
my $analysis;
until ($analysis)
{
    $tx = $ua->get($endpoint.$action_id.'/ ', $headers);
    eval { $analysis = $tx->res->json };
}

# The loop ends when we have response to the GET request
my $get_msg = $tx->res->message;
print "GET: '$get_msg'\n\n";

# In the GET response we have the result of the analysis
print "Analisys results:\n\n";
print encode_json $analysis;
print "\n";
}
```

## 8.2 Python

```
import requests, json

# User token, required for API access
oauth_token = 'tokenprovidedbybitext'

# API request data: Language and text to be analyzed
user_language = "eng"
user_text = "Your house is great"

print ("\nBITEXT API. POS Tagging endpoint. PYTHON sample code\n")
print ("-----\n\n")

# Building the POST request to pos tagging analysis endpoint
endpoint = "https://svc02.api.bitext.com/posttagging/"
headers = { "Authorization" : "bearer " + oauth_token, "Content-Type" : "application/json" }
params = { "language" : user_language, "text" : user_text }

# Sending the POST request
res = requests.post ( endpoint, headers=headers, data=json.dumps(params) )

# Processing the result of the POST request
post_result = json.loads(res.text).get('success')           # Success of the request
post_result_code = res.status_code                         # Error code, if applicable
post_msg = json.loads(res.text).get('message')           # Error message, if applicable
action_id = json.loads(res.text).get('resultid')         # Identifier to request the analysis results

print ( "POST: '" + post_msg + "'\n\n" )

# 401 is the error code corresponding to an invalid token
if ( post_result_code == 401 ):
    print ( "Your authentication token is not correct\n" )

if ( post_result ):
    print ( "Waiting for analysis results...\n\n" )

    # GET request loop, using the response identifier returned in the POST answer
    analysis = None
    while analysis == None:
        res = requests.get(endpoint + action_id + '/', headers=headers)

        if res.status_code == 200 :
            analysis = res.text
```

```
# The loop ends when we have response to the GET request
get_msg = res.reason
print ( "GET: '" + get_msg + "'\n\n" )

# In the GET response we have the result of the analysis
print ( "Analysis results:\n\n" )
print ( analysis )
print ( "\n" )
```



## 8.3 Ruby

```
require 'httpclient'
require 'json'

# User token, required for API access
oauth_token = 'tokenprovidedbybitext'

# API request data: Language and text to be analyzed
user_language = 'eng'
user_text = "Your house is great"

puts ''
puts 'BITEXT API. POS Tagging endpoint. Ruby sample code'
puts '-----'
puts ''

clnt = HTTPClient.new

# Building the POST request to pos tagging analysis endpoint
endpoint = "https://svc02.api.bitext.com/posttagging/"
headers = { "Authorization" => "bearer " + oauth_token, "Content-Type" => "application/json"}
params = {"language" => "" + user_language + "" , "text" => "" + user_text + ""}
.to_json

# Sending the POST request
res = clnt.post(endpoint,params,headers)

# Processing the result of the POST request
content = JSON.parse(res.content)

post_result_code = res.status # Error code, if applicable
action_id = content["resultid"] # Identifier to request the analysis results
post_msg = content["message"] # Error message, if applicable
post_result = content["success"] # Success of the request

puts 'POST: ' + post_msg
puts ''

# 401 is the error code corresponding to an invalid token
if ( post_result_code == 401 )
  puts 'Your authentication token is not correct'
end
```

```

if ( post_result )

    puts 'Waiting for analysis results...'
    puts ''

    # GET request loop, using the response identifier returned in the POST
answer
    analysis = ""

    while analysis == "" do

        res = clnt.get(endpoint + action_id + "/", {}, headers)
        if res.status == 200
            analysis = res.content
        elsif res.status != 202
            puts "GET: Error code (#{res.status}), message: " +
res.reason
                exit
        end
    end

    # The loop ends when we have response to the GET request
    get_msg = res.reason
    puts 'GET: ' + get_msg
    puts ''

    # In the GET response we have the result of the analysis
    puts 'Analysis results:'
    puts ''
    puts analysis
    puts ''
end

```

## 8.4 Java

```
import java.net.HttpURLConnection;
import java.net.URL;
import java.io.*;
import org.json.JSONObject;

public class codeSample {

    // User token, required for API access
    public static String oauth_token = "tokenprovidedbybitext";

    // API request data: Language and text to be analyzed
    public static String user_language = "eng";
    public static List user_text = "Your house is great";

    // Building the POST request to POS Tagging analysis endpoint
    public static String endpoint = "https://svc02.api.bitext.com/posttagging/";
    public static String textdata = "{\"language\":\"" + user_language + "\",\"text\":\"" + user_text + "\"}";

    public static void main(String[] args) throws Exception
    {
        System.out.println( "\nBITEXT API. Pos Tagging endpoint. JAVA
sample code" );
        System.out.println( "-----\n" );

        // Sending the POST request
        URL urlPOST = new URL(endpoint);
        HttpURLConnection connectionPOST = createAPIConnection("POST",urlPOST);
        DataOutputStream outputStream = new DataOutputStream(connectionPOST.getOutputStream());
        outputStream.write(textdata.getBytes("UTF8"));
        outputStream.flush();
        outputStream.close();

        // Processing the response code of the POST request
        switch (connectionPOST.getResponseCode())
        {
            case 201: // 201 is the code for succesful request processing
                break;
            case 401: // 401 is the error code corresponding to an invalid token
                System.out.println ( "Your authentication token is not correct" );
                System.exit(0);
            case 402: System.out.println ( "No contract found for that language" );
                System.exit(0);
        }
    }
}
```

```

        default:
            break;
    }

    // Processing the result of the POST request
    String sResponse1 = getAPIResponse(connectionPOST);
    JSONObject jsonResponse1 = new JSONObject(sResponse1);
    String action_id = jsonResponse1.getString("resultid"); // Identifier to request the analysis results
    String post_msg = jsonResponse1.getString("message");
    // Error message, if applicable
    boolean post_result = jsonResponse1.getBoolean("success");
    // Success of the request

    System.out.println ( "POST: " + post_msg + "\n");

    if (post_result)
    {
        System.out.println ( "Waiting for analysis results..
\n" );

        // GET request loop, using the response identifier returned in the POST answer
        // Ask for the result of the analysis launched before
        // Try until the analysis is ready and the API returns it

        URL urlGET = new URL(endpoint + action_id + "/");
        String sResponse2 = "";
        HttpURLConnection connectionGET = null;

        while (sResponse2 == "")
        {
            connectionGET = createAPIConnection("GET",urlGET);

            if (connectionGET.getResponseCode() == 200)
            {
                sResponse2 = getAPIResponse(connectionGET);
            }
        }

        // The loop ends when we have response to the GET request
        System.out.println ( "POST: " + connectionGET.getResponseMessage() + "\n");

        // In the GET response we have the result of the analysis
        System.out.println ( "Analysis results:\n" );
    }
}

```

```

        System.out.println(sResponse2); // Print it with spe
specified indentation
        System.out.println ( "" );
    }
}

    public static HttpURLConnection createAPIConnection(String method, URL
url) throws Exception
    {

        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod(method);
        conn.setRequestProperty("Content-Type", "application/json");
        conn.setRequestProperty("Authorization", "bearer " + oauth_token);

        if (method == "POST")
        {
            conn.setDoOutput(true);
            conn.setDoInput(true);
        }
        return conn;
    }

    public static String getAPIResponse(HttpURLConnection conn) throws Exception
    {

        String sResponse = "";
        BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInp
utStream()));
        String decodedString;

        while ((decodedString = in.readLine()) != null)
        {
            sResponse = sResponse+"\n"+decodedString;
        }
        sResponse = sResponse.trim();
        in.close();
        return sResponse;
    }
}

```