bitext when big data means big text

Technical Note
REST API Description

REST API description. Version 1.1. Jul 2017.

Document's information

Project	**BITEXT ENTITY EXTRACTION API**
Description	REST API description. Version 1.1. Jul 2017.
Author (s)	Luis Miguel Serrano; Luis Caloto
Customer / Dept.	
Last Mod. Date	July 12th, 2017
Issue	A
DOC_ID	

Table of contents

1 Purpose

This document describes how to use the Bitext Entity Extraction API.

2 Scope

This document applies to the integration and adaptation of the Bitext Entity Extraction 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

Entity:	*An individual item within a given text that specifies a specific instance of something in reality: person names, place names, dates, car plates, etc.*

4.2 Acronyms and Abbreviations

ASCII	***A**merican **S**tandard **C**ode for **I**nformation **I**nterchange* (character encoding set)
API	***A**pplication **P**rogramming **I**nterface*
SW	***S**oft**w**are*
CSV	***C**omma **S**eparated **V**alues* – (structured plain text format)
UTF-8	***UCS T**ransformation **F**ormat – 8 bits* (character encoding set)
UCS	***U**niversal **C**haracter **S**et* (character set standard)
ISO-8859-1	Character encoding set based on Latin character set for most Western European languages used mainly in Unix/Linux Operating Systems
Windows1252	Character encoding set based on Latin character set for most Western European languages used mainly in Windows Operating Systems

6 REST API

The Bitext Entity Extraction API can be reached by using the following endpoint:

```
https://svc02.api.bitext.com/entities/
```

For this API, the following languages are available: Dutch, English, French, German, Italian, Portuguese and Spanish.

You must specify the language code as parameter for requests:

Language	Code
Dutch	nld
English	eng
French	fra
German	deu
Italian	ita
Portuguese	por
Spanish	spa

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/entities/ -H "Authorization: bea
rer 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/entities/ -H "Authorization: bea
rer 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/entities/ -H "Authorization: bea
rer tokenprovidedbybitext" -H "Content-Type: application/json" --data '{"languag
e":"eng", "text":"Barack Obama is due to visit Scotland from April 20th"}'
```

This request is calling to the “entitiesanalysis” 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/entities/9a23759db4ae46d8bfbbab47
ac4dff1f/ -H "Authorization: bearer tokenprovidedbybitext" -H "Content-Type: app
lication/json"
```

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

```
{
  "resultid": "9a23759db4ae46d8bfbbab47ac4dff1f",
  "entitiesanalysis": [
    {
      "entity_norm": "Barack Obama",
      "entity": "Barack Obama",
      "type": "1"
```

```
    },
    {
      "entity_norm": "Scotland",
      "entity": "Scotland",
      "type": "3"
    },
    {
      "entity_norm": "20/04/",
      "entity": "April 20th",
      "type": "10"
    }
  ]
}
```

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 Entity Extraction Analysis

The Bitext Entity Extraction service provides the list of named entities present in a sentence. This service uses using syntactic and morphological analysis, apart from glossaries and dictionaries. For example, for the sentence “Barack Obama is due to visit Scotland from April 20th”, the Entity Extraction service will return the entities “Barack Obama” (person), “Scotland” (place) and “April 20th” (date).

7.1 POST Request

Request

- Endpoint: /entities/
- Protocol: HTTPS
- URL: https://svc02.api.bitext.com/entities/
- 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: /entities/
- Protocol: HTTPS
- URL: https://svc02.api.bitext.com/entities/: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": "...",
  "entitiesanalysis": [
    {
      "entity_norm": "...",
      "entity": "...",
      "type": "..."
```

```
    },
    {
      "entity_norm": "...",
      "entity": "...",
      "type": "..."
    },
    {
      "entity_norm": "...",
      "entity": "...",
      "type": "..."
    }
  ]
}
```

- 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)
- entity: the entity as it appears in the sentence. Example: "April 20th"
- entity_norm: the entity in its normalized form. Example: "20/04"
- type: the type of entity. Current entity types are:

0	Unknown/Not Assigned/Doubtful/Candidate
1	Person name
2	Car plate number
3	Place (City, Country, Region…)
4	Phone number
5	E-mail address
6	Company/Brand/Trade Mark
7	Organization
8	Web address/URL
9	IP address
10	Date (date only)
11	Hour
14	IBAN [Bank Account Number]
15	Money/currency amount
16	Address
17	Twitter hashtag [#XXX]
18	Twitter user [@XXX]
19	Other Alphanumeric
777	Reserved for internal use.

7.3 Example

POST Request

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

POST Response

```
{
  "success": true,
  "message": "Request accepted",
  "resultid": "12345123451234512345123451234512345ab"
}
```

GET Request

```
curl -s -X GET --url https://svc02.api.bitext.com/entities/123456789abcdefg12345
6789abcdefg/ -H "Authorization: bearer tokenprovidedbybitext" -H "Content-Type:
application/json"
```

GET Response

```
{
  "resultid": "1b84e78a8655448db6b1730927fba1a2",
  "entitiesanalysis": [
    {
      "entity_norm": "Barack Obama",
      "entity": "Barack Obama",
      "type": "1"
    },
    {
      "entity_norm": "Scotland",
      "entity": "Scotland",
      "type": "3"
    },
    {
      "entity_norm": "20/04/",
      "entity": "April 20th",
      "type": "10"
    }
  ]
}
```

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 = "Barack Obama is due to visit Scotland from April 20th";

print "\nBITEXT API. ENTITY EXTRACTION endpoint. PERL sample code\n";
print "-----------------------------------------------\n\n";

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

# Building the POST request to Entity Extraction analysis endpoint
my $endpoint = "https://svc02.api.bitext.com/entities/";
my $headers = { Authorization => "bearer $oauth_token", 'Content-Type' => 'appli
cation/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 applica
ble
my $post_msg = $tx->res->json->{message};              # Error message, if appl
icable
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 analisis results...\n\n";

         # GET request loop, using the response identifier returned in the POST
 answer
         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 = "Barack Obama is due to visit Scotland from April 20th"

print ("\nBITEXT API. ENTITY EXTRACTION endpoint. PYTHON sample code\n")
print ("-------------------------------------------------\n\n")

# Building the POST request to Entity Extraction analysis endpoint
endpoint = "https://svc02.api.bitext.com/entities/"
headers = { "Authorization" : "bearer " + oauth_token, "Content-Type" : "applica
tion/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 applica
ble
post_msg = json.loads(res.text).get('message')         # Error message, if appl
icable
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 analisis 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=header
s);
                  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 ( "Analisys 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 = 'end'
user_text = "Barack Obama is due to visit Scotland from April 20th"

puts ''
puts 'BITEXT API. ENTITY EXTRACTION endpoint. Ruby sample code'
puts '-----------------------------------------------'
puts ''

clnt = HTTPClient.new

# Building the POST request to Entity Extraction analysis endpoint
endpoint = "https://svc02.api.bitext.com/entities/"
headers = { "Authorization" => "bearer " + oauth_token, "Content-Type" => "appli
cation/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 applica
ble
action_id = content["resultid"]                        # Identifier to request
the analysis results
post_msg = content["message"]                          # Error message, if appl
icable
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 analisis 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 'Analisys 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 = "Barack Obama is due to visit Scotland
from April 20th";

         // Building the POST request to ENTITY EXTRACTION analysis endpoint
    public static String endpoint = "https://svc02.api.bitext.com/entities/";
    public static String textdata = "{\"language\":\"" + user_language + "\",\"t
ext\":\"" + user_text + "\"}";

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

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

                   // Processing the response code of the POST request
                   switch (connectionPOST.getResponseCode())
                   {
            case 201:  // 201 is the code for succesful request processing
                       break;
                                                  // 401 is the error code corre
sponding to an invalid token
            case 401:  System.out.println ( "Your authentication token is not co
rrect" );
                       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 jResponse1 = new JSONObject(sResponse1);
        String action_id = jResponse1.getString("resultid");              // Ide
ntifier to request the analysis results
                  String post_msg = jResponse1.getString("message");
                  // Error message, if applicable
                  boolean post_result = jResponse1.getBoolean("success");
         // Success of the request

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

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

                           // GET request loop, using the response identifier r
eturned in the POST answer
                           // Ask for the result of the analysis launched befor
e
                           // Try until the analysis is ready and the API retur
ns it

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

                           while (sResponse2 == "")
                           {
                                    connectionGET = createAPIConnection("GET",u
rlGET);
                                    if (connectionGET.getResponseCode() == 200)
                                    {
                                             sResponse2 = getAPIResponse(conne
ctionGET);
                                    }
                           }

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

                           // In the GET response we have the result of the ana
lysis
                           System.out.println ( "Analisys results:\n" );
```

```
                            System.out.println(sResponse2); // Print it with spe
cified 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;
    }
}
```