Jeremy REST Service

The Jeremy REST service is a way for developers to interface with our Jeremy API without any included library in their code. It also allows for any updates that we make to the API itself to be automatically be used in your program. To start using this service you must have an account on the Jeremy Web service, a basic understanding of REST services and understand how to call a REST service where you need it.

REST URLs

The URLs are where you are sending your request to and the Jeremy REST service has two of them. These start with ***"http://localhost:8080/Web_Server/rest/restconverter"*** and you can add ***"/xml"*** to get an XML file back or ***"/json"*** to get a JSON file back.

Input requirements

The Jeremy REST service consumes certain input of the content type **"application/x-www-form-urlencoded".** Each value that is put in here is used as a key-value pair where some are required and some are not.

Key	Value Type	Required	Description
csvData	String	Yes	The Comma separated data you wish to convert. See "delimiter " field to change separated value
encodedEmail	String	Yes	Your encoded email. See "Encoding Standard" for what to use here
publicAPIKey	String	Yes	Your public API key that is assigned to your account
tableName	String	Yes	The table name you wish to use
useFirstLineAsHeaders	Boolean	No, Defaults to false	If you would like to make the first line of your data column headers
dateFormat	String	No, Defaults to 'dd/MM/yyyy'	The REGEX date format you wish to use to convert your date data
delimiter	String	No, Defaults to ','	What your columns are separated by in the data you have provided

Encoding Standard

To use this service you need to provide your encoded email. Here are steps on how to encode it to fit the needs of our system:

1. Concatenate your lowercased email and private key together.
2. Get the bytes from your concatenated email and private key string.
3. Encode that string via HmaxSHA1

4. Encode resulting byte array via Base64

You can now use the Trimmed Base64 string as your encoded email. You can model your client side hashSecrets method on the following:

```
public static String hashSecrets(String email, String privateKey) throws
NoSuchAlgorithmException, InvalidKeyException {
             // Create an "Sting of entropy" which is just a String containing
the email + private key
             String entropyString = email.toLowerCase() + privateKey;
           SecretKey secretKey = null;

           // Convert the entropyString to a byte array to use it in the
algorithm.
           byte[] keyBytes = entropyString.getBytes();
           // Encode with HmacSHA1
           secretKey = new SecretKeySpec(keyBytes, "HmacSHA1");

           // Setup the crpyto stuff
           Mac mac = Mac.getInstance("HmacSHA1");
           mac.init(secretKey);
           byte[] rawHmac = mac.doFinal(entropyString.getBytes());

           // Return a Base64 representation of the string encoded in HmacSHA1
           return new String(Base64.encodeBase64(rawHmac)).trim();
      }
```

And implement the client request as in:

```
private static void postTest(){
             //create access to the client
             Client client = ClientBuilder.newClient();
             WebTarget jsonTarget =
client.target(getBaseURI()).path("rest").path("restconverter").path("json");

             //Create form structure to send to the server
             Form form = new Form();
             form.param("csvData", getTestDataFileString());
             form.param("useFirstLineAsHeaders", "true");
             form.param("publicAPIKey", publicAPIKey);
             try {
                    String encodedEmail = hashSecrets(email, privateAPIKey);
                    form.param("encodedEmail", encodedEmail);
             } catch (Exception e){
                    e.printStackTrace();
                    System.exit(0);
             }
             form.param("tableName", "Apples");
             form.param("delimiter", ",");

             //send to the server
             Response jsonOutput =
jsonTarget.request(MediaType.TEXT_PLAIN).post(Entity.entity(form,
MediaType.APPLICATION_FORM_URLENCODED), Response.class);
      }
```

Output Results

The REST service has a number of different possible results that will be returned when something goes wrong or even if everything goes well. Here is the list of them

Code	Description
201	Everything went fine. Returns converted data.
403	"Hashed email not correct. Please use Base64 Encoding."
403	"Not a valid API key"
403	"Encoded email is empty"
406	"Please specify a table name"
406	"Please supply CSV data"
500	"Failed creation: Internal Error"
500	"Could not link Public API key with account"