Facebook Report

YiLin Liu

March 3, 2017

Quick Introduction To facebook API:

Structure:

The facebook Graph API (API used to access general facebook data) is in the form of a *graph*:

- *nodes:* Nodes are like Users, page, groups, comments etc. Each node has its' own unique ID that which is used to access itself via the Graph API.
- *edges:* The connections that leads from one node to another are called edges. For example: the cover photo of a user, the comments and posts on a user's timeline. Edges are identified by a name.
- *fields:* Information about the nodes are called fields,like the names, Id, birthdays of a user.Fields are identified by a name.

Request Format:

- The *API request* is in the format of a standard HTTP request. Methods like **GET,POST, DELETE, etc.** can be used to retrieve and modify information of facebook nodes through the API granted the correct access token is provided.
- Format of a request:
  `GET graph.facebook.com /{node-id}/{edge-name}?fields = {first-level fields}{Second-level fields}`.
  - The levels of the fields form a pipeline where the fields of the first level fields can be accessed as the second level fields.
- An OAuth access token is required to access any information, Rfacebook provides functionalities to handle tokens.

Pagination:

- To manage the amount of information returned per API Call, facebook divides the data using several pagination techniques (Cursors-based, Time-based and Offset-based).
- Most commonly used and the only ones encountered during this report are the curosr based paginations.
- The information for pagination provides a *next* token which is a HTTP GET request for the next page of data and a *previous* token which is a HTTP GET request for the previous page of data.
  - If the provided paging request call returns *NULL*, the end/very beginning of the data is reached.

Rate Limit:

- The type of rate limit dealt with mainly in this report is **Application-Level Rate Limiting**. Only the application access key was used. Access Tokens
- The facebook API states that the limit for the application is **200 calls per hour per user in aggregate.**
- However not all API calls are subjected to the rate limit.
- There is a tool that monitors the application's API calls and it is located under the dashboard of the application. Test Application. The user would need to be logged in.

Other Facebook API:

- There is **not** a paid Facebook Graph API.
- For commercial purposes, Facebook has another API called Marketing API which is used to manage ad campaigns.
- Another advertisment based API is the atlas API. From a basic google search. This page says that the atlas API uses identity of each user to target advertisment.

Rfacebook Package:

- Rfacebook

Summary of Findings For Rfacebook:

Advantages: * Clear documentation. * High level of Abstraction. * Good variety of functions that can provide data from facebook API. * The source code provides a structure to parse any arbitary facbook API response in JSON.

Disadvantages: * If an error is thrown by the facebook API, the search stops. * Many functions are no longer supported due to the updated facebook API. * Limited range of fields and edges, does not support the extraction of any arbitrary fields and edges.

List of Deprecated Functions:

- *getCheckins:* deprecated.
- *getFQLS:* deprecated.
- *getFriends:* only friends who uses the applications.
- *getNetwork:* only applicable to users of the application.
- *searchFacebook:* deprecated.

Setting Up Libraries

```
    ##Loading Libraries needed for the rest of the report:

    library(jsonlite)
```

```
## Warning: package 'jsonlite' was built under R version 3.3.3
```

```
    library(knitr)
    library(Rfacebook)
    knitr::opts_chunk$set(error=T,warning=TRUE)

    rm(list = ls())
```

Creating Test Account:

- *First name:* API-Testing
- *Last name:* nrc
- *Email:* NRC.API.Testing@gmail.com
- *Password:* NRCTesting123456!
- *Birthday:* July 1st 1997
- *Gender:* Males

Creating token:

- Normally, a temporary access token normally has a 2-hour expiration time. A long-lived token (OAuth token) can be used for longer access. The following outlines how to obtain a long-lived token:
  - ___The *fbOAuth* function documentation in Rfacebook outlines the steps.
  - Mining facebook Data Using R & facebook API! also provides steps to creating a long-lived token.

```
    app_Id = 1353820787971442
    app_Secret = "4841ab73f4f68960ebbf37e5705e2610"

    ## The following shows the code that calls the fbOAuth function
    ## and creates the OAuth token,
    ## it has been commented once executed and the result
    ## is saved in a file
    ## called "my_OAuth.txt" Please run the following code and follow
    ## the instructions printed in the
    ## console in case the file does not exist.

   # token = fbOAuth(app_id = app_Id,app_secret = app_Secret,extended_permissions = TRUE)
   # if(!file.exists("My_OAuth"))
   #  {
   #      file.create("My_OAuth.txt")
   #      save(token, file= "My_OAuth.txt")
   #  }

load("My_OAuth.txt")
   rm(list = setdiff(ls(),c("token","FbObjectId") ))
```

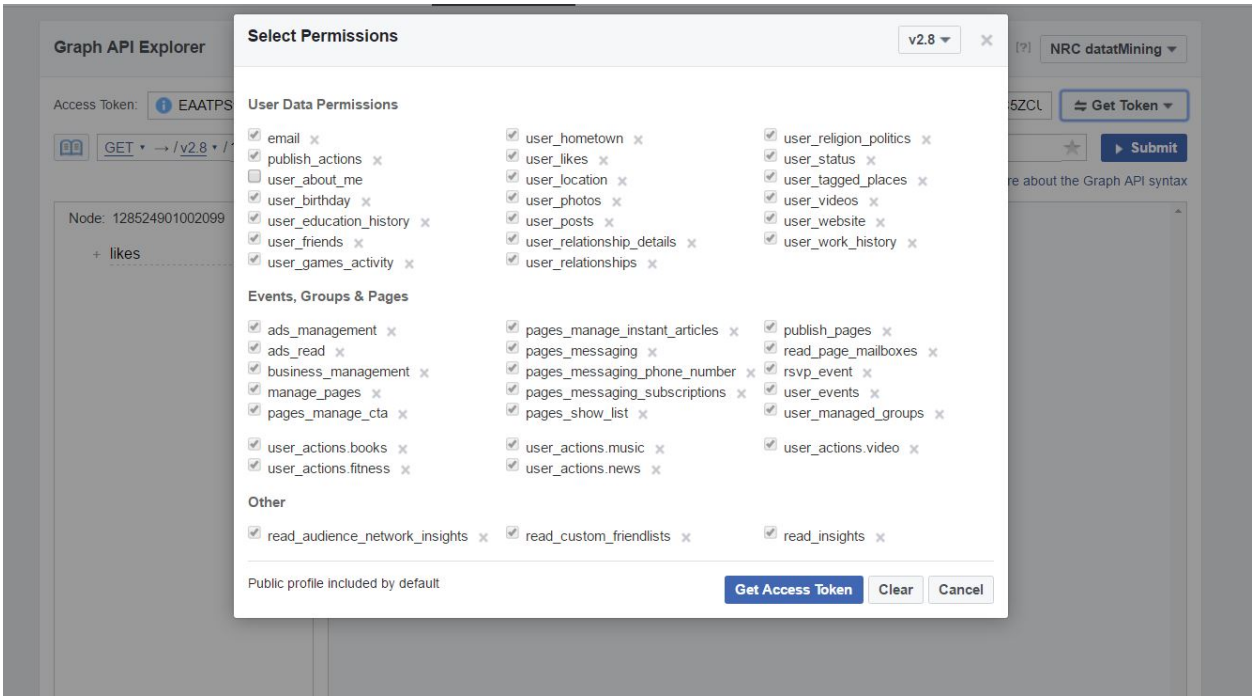


Figure 1: Extended Permissions

- The following pictures indicates where to navigate to set the extended permissions.

Exploring Rfacebook:

Location:

GOAL: To try and explore what the facebook API offer regarding extracting location information.

Exploring facebook API for Nodes that Contain Location Information:

Nodes	Description
Albums.	Contains location field. Often returns null
Comments	No location fields.
Events	Usually includes location information under the place field with a valid location.
Pages	Contains a location field, often yields no useful response.
Posts	Contains a place field, but usually returns NULL when requested.
Photos	Contains a place field for location.

---

callAPI Function:

- The "callAPI" function retrieves response for any valid facebook API requests.
- The main benefit is that the user can pass in any arbitrary API request. ex: _https://graph.facebook.

```
##
## $id
## [1] "128524901002099"
```

```
    ##  Adding fields to the URL to gain specific information (birthday, age_range)
    ##  no spaces anywhere within the URL should be included
    my_Specific_Information_Request = "https://graph.facebook.com/me?fields=birthday,age_range"
    my_Specific_Data = callAPI(my_Specific_Information_Request,token)

    print(my_Specific_Data)
```

```
## $birthday
## [1] "07/01/1997"
##
## $age_range
## $age_range$max
## [1] 20
##
## $age_range$min
## [1] 18
##
##
## $id
## [1] "128524901002099"
```

Id Function:

- The goal is to provide the object ID to access basic nodes. A simple function is made to extract that information. Not valid for complext URLs such as the ones for posts, comments, pictures etc.
- Test cases (all cases are behaving as expected)

Exploring *getPage,getPost,getCommentReplies*:

GOAL: To extract information regarding comments on a public facebook webpage

1: Starting with getting the post_Ids (mainly post Ids) using _getPage_

```
    require(Rfacebook)

    ##  the getPage function returns a matrix, and one of the
    ##  columns is the "postId"
    BBC_URL = "https://www.facebook.com/bbcnews"
    load("My_OAuth.txt")
    BBC_Id = FbObjectId(BBC_URL,token)
    print(BBC_Id)
```

```
## [1] "228735667216"
```

```
    ## retrieving page information using
    ## getPage function
    page_Information = getPage(BBC_Id,token,n=2)
```

```
## 2 posts
```

```
    names(page_Information)
```

```
##  [1] "from_id"        "from_name"      "message"        "created_time"
##  [5] "type"           "link"           "id"             "story"
##  [9] "likes_count"    "comments_count" "shares_count"
```

```
    ## the post_Ids
    post_Ids = page_Information$id
```

2: Getting information about each post using *getPost* (mainly interested in comment ids)

```
    ##  trying to see getPost can be vectorized,
    ##  it appears to be working through the use of lapply

    ## getting all the post information with
    ## getPost function
    post_Information = lapply(post_Ids,getPost,n=100,token=token)

   test <- post_Information[[2]]

    names(test)
```

```
## [1] "post"     "likes"    "comments"
```

```
    names(test$post)
```

```
##  [1] "from_id"        "from_name"      "message"        "created_time"
##  [5] "type"           "link"           "id"             "likes_count"
##  [9] "comments_count" "shares_count"
```

```
    names(test$likes)
```

```
## [1] "from_name" "from_id"
```

```
    names(test$comments)
```

```
## [1] "from_id"        "from_name"      "message"        "created_time"
## [5] "likes_count"    "comments_count" "id"
```

3: Using the *getcommentReplies* to further investigate each reply to a comment.

```
    ## Extracting comment_ids

    comments = unlist( lapply(post_Information, function(x) {x["comments"]}), recursive = F)
    comments_Id = unlist(comments[[1]]["id"])


    #Extracting the comment replies to each posts' comments.

    ## First test the getCommentReplies with a single comment_Id:
    single_Comment_Reply = getCommentReplies(comments_Id[[1]],token=token)
    names(single_Comment_Reply)
```

```
## [1] "comment" "replies"
```

```
    names(single_Comment_Reply$replies)
```

```
## [1] "from_id"      "from_name"    "message"      "created_time"
## [5] "likes_count"  "id"
```

```
    ##Now for all of the comment Ids via lapply.

    comment_Replies = lapply(comments_Id, function(x) getCommentReplies(x,token = token))

    length(comment_Replies)
```

```
## [1] 100
```

Exploring *searchGroup & getGroup* functions:

- ***GOAL: To extract information regarding a public facebook group: couponboutique using the getGroup function***

1: Suppose the ID is not known, or not certain. *searchGroup()* can be used to extract the group_Id:

```
    load("My_OAuth.txt")
    group_Name = URLencode("The Coupon Boutique")
    group_Id = searchGroup(group_Name,token=token)
    print(group_Id)
```

```
##                                name privacy               id
## 1 The Coupon Boutique - Public Group    OPEN 1696255650601823
## 2                The Coupon Boutique  CLOSED 1504395996479976
## 3               The Coupon Boutique2    OPEN 1580411485549988
## 4         Alexis' Couponing Boutique  CLOSED  279585609058254
```

```
    ##  Also works when URLencoding the search string
    group_Name_URL = URLencode("The Coupon Boutique")
    group_Id2 = searchGroup(group_Name_URL,token=token)
    print(group_Id2)
```

```
##                                name privacy               id
## 1 The Coupon Boutique - Public Group    OPEN 1696255650601823
## 2                The Coupon Boutique  CLOSED 1504395996479976
## 3               The Coupon Boutique2    OPEN 1580411485549988
## 4         Alexis' Couponing Boutique  CLOSED  279585609058254
```

2: Extracting information using *getGroups*

```
    start_Time = as.numeric(as.POSIXct("2016-12-01"))
    id = group_Id2$id[[1]]

    ##  This id was manually verfied to be
    ##  the group that is listed at the top
    print(id)
```

```
## [1] "1696255650601823"
```

```
    ## NOTE: the "groupId" is different from the "pageId"
    ## where the group is posting and hosting.

    #NOTE: the default number of posts to return is 25.
    group_posts = getGroup(id,token=token)
```

```
## 25 posts
```

```
    class(group_posts)
```

```
## [1] "data.frame"
```

```
    names(group_posts)
```

```
##  [1] "from_id"        "from_name"      "message"        "created_time"
##  [5] "type"           "link"           "id"             "story"
##  [9] "likes_count"    "comments_count" "shares_count"
```

```
    rm(list = setdiff(ls(),c("folder_Path","token","FbObjectId") ))
```

Exploring *searchPage*:

- This method works as shown in the documentation.

```
    keyword ="Apples"
    data = searchPages(keyword,token, n=100)
```

```
## 100 pages
```

```
## Warning in vect[notnulls] <- unlist(lapply(lst, function(x) x[[field]])):
## number of items to replace is not a multiple of replacement length
```

```
    class(data)
```

```
## [1] "data.frame"
```

```
   names(data)
```

```
##  [1] "id"                  "about"               "category"
##  [4] "description"         "general_info"        "likes"
##  [7] "link"                "city"                "state"
## [10] "country"             "latitude"            "longitude"
## [13] "name"                "talking_about_count" "username"
## [16] "website"
```

```
   rm(list = setdiff(ls(),c("folder_Path","token","FbObjectId") ))
```

Trying *getLikes*:

- ***GOAL:The documentation stated that it could provide information about a user or a page's likes given the id.***
- ***Does not work on users but work on pages.***

Search API:

- Allows searches on public information.
- Search Documentation
- **This API provides the same functionality as facebook's Search Bar.**

- Query example: `https:graph.facebook.com/search?q={string to search for}&type={type name}&{fields of type}`

Mini Project1:

Description:

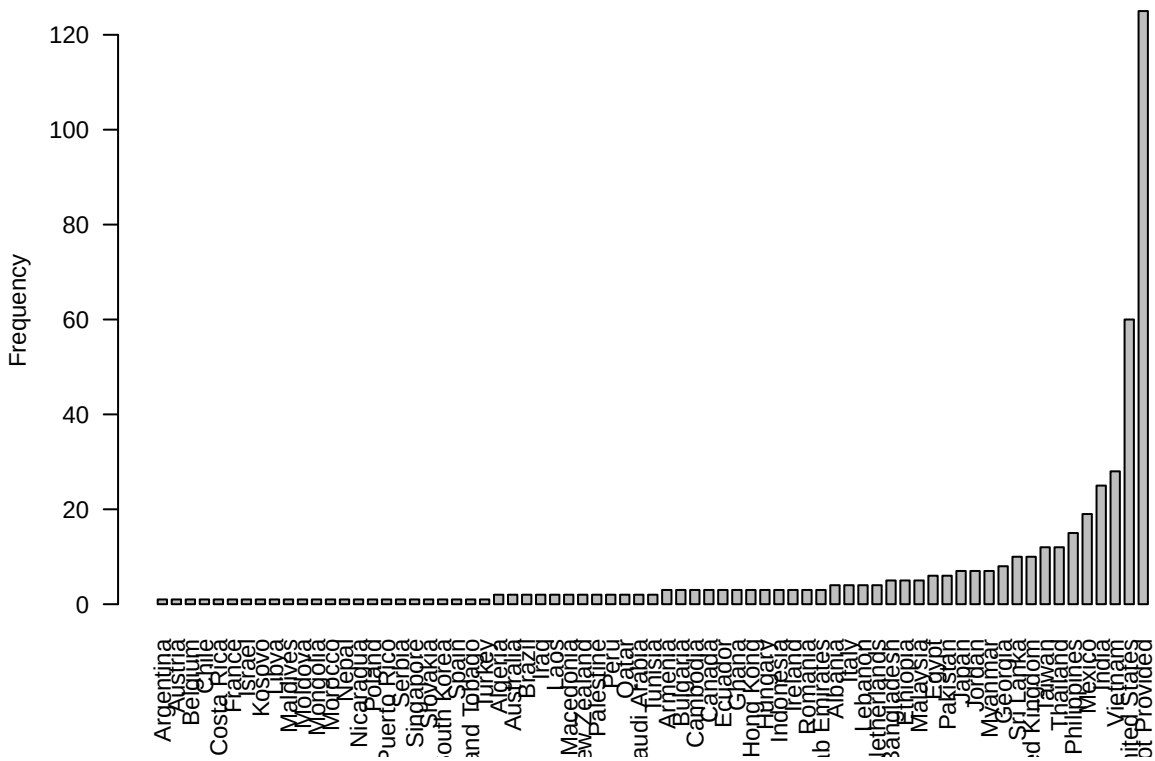
1: To scrape the facebook API for all web pages that contains a series of keywords. Up to 2000 pages, then graph distribution based on location.

Code:

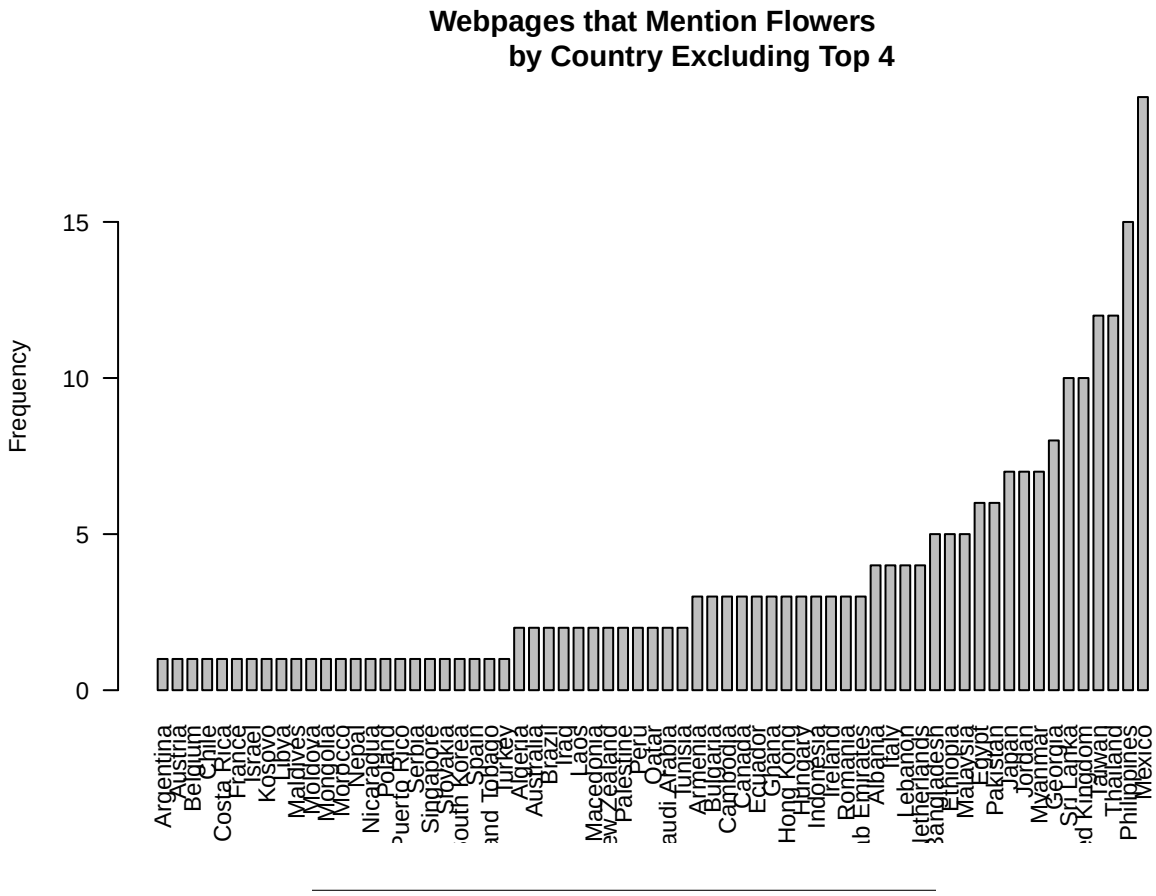
```
    rm(list = ls())

    #Loading libraries
    library(Rfacebook)

    # setting up authorization token
    ## refer to above section for details
    load("My_OAuth.txt")

    keywords = c("flower")

    ##Note: there might not be the specified number of pages
    ##that will be returned as the result.
    webpage= searchPages(keywords, token=token,n=2000)
```

```
## 200 pages 400  472
```

```
    webpage_Location = webpage$country
    webpage_Location[is.na(webpage_Location)] = "Not Provided"

    #presenting information:
    frequency_table = table(webpage_Location)
    par(cex=0.75)
    barplot(sort(frequency_table),las=2,main="Webpages that Mention Flowers by Country"
            ,ylab="Frequency",space=c(0.5))
```




```
## Replotting graphs to exclude the top 4 Countries.
frequency_table1= frequency_table
for(i in 1:4)
{
    frequency_table1= subset(frequency_table1,
                             frequency_table1 < max(frequency_table1))
}
barplot(sort(frequency_table1),las=2,main="Webpages that Mention Flowers
        by Country Excluding Top 4",ylab="Frequency",space=c(0.5))
```



MiniProject 2:

Description:

_To find the place most posted on a travel website from 500 posts. The main point is trying to illustrate the structure of the response and processing to a dataframe.

- note: sometimes there is no location tag, could consider searching for it in the text of the comment.

Code:

Gathering RawData:

```
rm(list = ls())
website_URL = "https://www.facebook.com/thousandamazingplacesonearth/"
load("My_OAuth.txt")

website_Id = "437841726243764"

##  Making the query. THe query was
##  first tested this out on the Graph API Explorer
```

```
##  Note: The limit for number of posts is capped
##  at 100 for a page of response.
##  Additional responses needs to be retrieved
##  via cursors.

query= paste("https://graph.facebook.com",website_Id,
             "?fields=posts.limit(100){place}",sep="/")

#Pulling RawData
raw_Data= callAPI(query,token)
```

Pesudocode For Parsing:

NOTE: This process is based on similar procedure as the source code of the Rfacebook Package. No credit is taken for the idea behind this parsing. The structure of the Rfacebook source code was studied and parts of it was adapted. The functions that was studied are the getPage function, the *pageToDF & UnlistWithNA* functions under the Utils.R package

- It is strongly recommended to use the function *UnlistWithNA* (unavailable for direct access in the Rfacebook Package) under the Utils.R package for parsing of queries not supported by the methods of the Rfacebook. *UnlistWithNa* function includes pre-made parsing function for many possible fields. Furthermore, the general structure of this function can be adapted to parse JSON formatted information.___

Parsing and Processing information.

```
num_Posts = 500
post_Limit = 100

#   navigating to list of posts data.
query_Data = raw_Data$posts$data

#   extracting information and formatting.
#   This process is mostly done by the UnlistWithNA function.

UnlistWithNA_Copy = function(field, list)
{
    ## produce a list of NAs.
    complete = rep(NA, length(list))

    if(length(field)==1)
    {
        ## produces a vector indicating which indices are not null.
        notNull = unlist(lapply(list,function(x) !is.null(x[[field]])))

        ##  Combine the information within the fields as well as
        ##  the NA_List to
        ##  form the complete and correct list.
        ##  NOTE: differences between [[]] selects the content and []
        ##  selects the corresponding container and the content

        complete[notNull]= unlist(lapply(list, function(x) x=x[[field]]))
```

```
    }

    ##  Multiple level of fields to parse through
    if(length(field)==2)
    {
        ##similar logic as the above.
        notNull=unlist(lapply(list,function(x)!
                                is.null(x[[field[1]]][[field[2]]])))
        complete[notNull]= unlist(lapply(list,function(x)
            x=x[[field[1]]][[field[2]]]))

    }
    if (length(field)==3){
        notnull <- unlist(lapply(list, function(x)
        !is.null(x[[field[1]]][[field[2]]][[field[3]]])))

        complete[notnull] <- unlist(lapply(list[notnull],
                           function(x) x = x[[field[1]]][[field[2]]][[field[3]]] ))
    }

     ## Note: for lists, data.frame corece it as a row. While for vectors,
     ## each vector is a column. Thus in order for each field to
     ## be its own column,
     ## the list needs to be a vector
    return(as.vector(complete))

}


## Takes a list of data and extract the fields of interest
## and organize into a data frame.
OrganizingDF = function(list)
{
    # Constructing lists corresponding to the desired fields using the above
    # function.

    ##Similar Warning appeared that stated multiple of replacement length when
    ##the package is called orginally.

        post_Id=UnlistWithNA_Copy(c("id"), list)
        place_Id=UnlistWithNA_Copy(c("place","id"),list)
        place_Name=UnlistWithNA_Copy(c("place","name"),list)
        city=UnlistWithNA_Copy(c("place","location","city"),list)
        country=UnlistWithNA_Copy(c("place","location","country"),list)
        latitude=UnlistWithNA_Copy(c("place","location","latitude"),list)
        longitude=UnlistWithNA_Copy(c("place","location","longitude"),list)
        street=UnlistWithNA_Copy(c("place","location","street"),list)

    df = data.frame(post_Id, place_Id,place_Name,city,country,
                    latitude,longitude,street,stringsAsFactors = F)

    return(df)
}
```

```
## Continuing on with Data Processing:
## The query information is organized into a data frame.

query_DF = OrganizingDF(query_Data)


##  Moving on to organizing the paging informations. This can be done in a loop.
##  Depending the number of posts desired, a custom function can be made
##  to calculate how many pages of response are required.

## By modifying the ".limit" modifier in the paging cursor,
##  you can specify the number of posts to return per page of response.
##  In this case, the required number of posts is 500
##  a loop will be made to navigaet the cursors 4 times.

##  The format of the paging token location varies: For the initial query,
##  it is under raw_Data$posts$paging[["next"]]. For any subsequent navigations,
##  it is under raw_Data$paging[["next"]]

paging_Url = raw_Data$posts$paging[["next"]]
for(i in (1:4))
{
    #formatting paging_Url to extract 100 posts (post limit) each time
    paging_Url = gsub(pattern="place&limit=25", replacement =
                          paste("place&limit=",post_Limit,sep=""), x=paging_Url)


    # Extracting the raw_Data from the new paging_Url
    raw_Data_Paging = callAPI(paging_Url,token)

    #formatting the data using the above functions into a data frame.
    response_DF = OrganizingDF(raw_Data_Paging$data)

    #Formating final output:
    query_DF = rbind(query_DF,response_DF)

    #Navigating to next paging cursor
    paging_Url = raw_Data_Paging$paging[["next"]]
}

dim(query_DF)
```

```
## [1] 500   8
```

```
# Now the matrix is organized, time to get it into the final output format.

final_Out = list(query_DF, raw_Data_Paging$paging)
```

```
#finding most visited location:

location = query_DF$country
freq= table(location)
freq=sort(freq)
print(freq[length(freq)])
```

```
## India
##    32
```