Paper 121-2012

Accessing and Extracting Data from the Internet Using SAS®

George Zhu, Sunita Ghosh, Alberta Health Services - Cancer Care

ABSTRACT

This paper shows you how to use SAS® to automatically access web pages from Internet and extract useful information from the web pages. We first review some basic elements of Internet accessing, then review the accessing and extraction tools provided by Base SAS. When the SAS tools are not enough or too complicated to use, we show how you can use external programs (cURL and Perl/LWP) and integrate the results from external programs into a SAS data set. Finally, we present 4 different examples to illustrate how to use these tools to deal with different levels of complexity for web accessing and data extraction.

INTRODUCTION

The Internet provides huge amount of data and information, which we can browse and download them using a browser. What if we want to download and extract the information automatically using SAS and integrate the information with SAS data sets? For example, can we extract the real time stock index using SAS? or can we monitor the career websites of some companies so that whenever a suitable opportunity is posted, you will be notified? Because SAS is very powerful in data management, data mining and analytics, we would like to combine the power of SAS and the gold mine of data from the Internet. In this paper, we will explain how you can use the tools provided by SAS to dig out the useful information from the Internet and integrate the information with existing data in the SAS data sets. We will first review some basic elements about Internet, and review the SAS tools and other tools for web accessing and data extraction. Finally we provide examples to indicate how you can use these tools to automatically and efficiently obtain information from different websites.

ELEMENTS OF INTERNET

To understand how to access Internet using SAS, we need to know some basic elements about Internet. More specifically, we need to know where is the information located (the Universal Resource Locater, URL), how the information is organized (the Hypertext Markup Language, HTML), how to communicate with the server (the Request methods, GET or POST), and some more advanced elements, like HTTP headers, which include the dealing with cookies, user agent, response status code, redirects, etc. Helf (2005) provided more detailed and yet clear explanation on most of these topics. In this section, we will only focus on URL, HTML and Request methods.

URL

A Universal Resource Locator (URL) is the character string or the IP address that you type in the address bar in the web browser. It identifies the location of the web server and the target webpage to be fetched by the web browser. For example, when you want to browse the webpage for the Edmonton SAS User Group (*eSUG*) meeting, you can type in the following URL to the address bar in the web browser:

```
http://www.sas.com/offices/NA/canada/en/edmonton.html
```

and if you want to get a Google map of Edmonton, you can type in the URL as:

```
http://maps.google.com/maps?q=edmonton
```

Each URL consists of 3 parts. The first part is the transfer protocol, such as HTTP, HTTPS, FTP, etc. The second part is the host name or the IP address. In the above examples, the host names are *www.sas.com* and *maps.google.com*, respectively. The third part is the location of the webpage. In the first example, the webpage is

located in the folder of */offices/NA/canada/en/*, and the file name of the webpage is *edmonton.html*. This webpage is physically located in the SAS web server. This is called the static webpage. On the other hand, in the second example, the webpage is specified by *maps?q=edmonton*. This is not physical webpage, it is called the dynamic webpage. A dynamic webpage is consisted with a function name (a computer program in the web server) and the parameter/value pairs. In this example, there is a computer program called *maps*, which accepts one parameter (*q*), and in that situation, we supply the value "*edmonton*" to parameter *q*. Upon receiving this value, the maps function on the Google Maps server will be executed and generates a webpage and then sends it back to the client, and we can see the map of Edmonton in our browser.

URLs can not contain space or any special characters (such as #,; etc.). If these special characters need to be included in the URL, they need to be *encoded*. You can use SAS function *URLENCODE* to do this. The other function *URLDECODE* can do the opposite. Also, another pair of SAS functions, *HTMLENCODE* and *HTMLDECODE*, can be use to encode/decode the content (html) file.

For dynamic webpages, if there are more than two parameter/value pairs, an ampsand sign (&) is used to separate the pairs. Since & is reserved for referencing macro variable in SAS, the URLs need to be quoted (use the macro quoting functions, such as *%STR*, *%QUOTE*, *%SUPERQ*, etc.).

GET AND POST METHODS

The request method defines how the client (your browser) send the URL to the server. The two most used request methods are GET and POST. By GET method, the request string parameter is included in the URL, while by POST method, the request parameters is sent through content section in the request header. In a web browser, you can see all the request parameters in the address bar for the GET method, but for the POST, you won't be able to see it: You may click a link and go to next webpage, but the URL in the address bar does not change at all. The POST method is used to send large amount of information, or sensitive data (such as password) to the server. The request method usually is specified in the html code with the "*form*" tag (see next). The *form* tag has an attribute called "*method*". The default method is GET. When it is specified as: "*method=post*", then the request will be sent using the POST method. So we can examine the html code of the current webpage, search for the *form* tag and find out what request method it uses in order to go to other webpages.

HTML

The response from the server is usually in the form of HTML (Hypertext Markup Language) codes. If you are using web browser, you can right-click and select *View page source* to see the webpage in HTML codes. HTML file is a text file. It contains various of tags, which are used by the web browser to determine how to organize and display the information in the screen.

Examples of tags (normally come in pairs: the beginning and the ending) are:

- `<html> ... </html>`: beginning and ending of a html document. Every html file must have this tag
- `<a href=...>`: for specifying a link (an URL)
- `<div id=...>`: for identifying a division or a section in an html document
- `<table>, <th>, <tr>, <td>`: identify the table, table head, table row, table data
- `<form ...method="post">, <input..name=** value=**>`: specify what parameters (name) and values are needed to be sent to the server, and what method to send the request (get or post)

Because HTML file is a text file, it can be processed and read by SAS. We can read the HTML file into a SAS data set, store each line in one record in a string variable. The *INFILE* statement can be used to read the HTML into a data set. Since the length of the lines in HTML file is not constant, we should use *varying* length to read the lines. Also, be aware that SAS has a length limit of 32767 - characters after this limit in a line can't be read into SAS, and it is not unusual to have lines longer than 32767 in a HTML file.

Once the HTML file is read into a data set, you can use many SAS string functions to extract the information contained in the HTML file. Here we can use the HTML tags to help us locate the target information. We'll show more detail on data extraction in the examples.

WEB ACCESSING AND DATA EXTRACTION TOOLS

To access a webpage, we usually use a web browser, such as Internet Explorer, Firefox, Google Chrome, etc. Web browsers have much more functionalities, but they are for human interaction, such as browsing, and clicking a link to next page, etc., they are not for automatic data extraction. On the other hand, if you are a programmer, you might know the tools such as the cURL command line program, the LWP (library for WWW in Perl) package. Unlike web browsers, these programs are for automatic internet accessing. Although they don't have the same functionalities as web browsers, for example, they can't handle Javascript codes, by using appropriate options they can mimac most web browser functions. While cURL is mainly for web accessing, with the powerful Perl Regular Expression functions for information extraction, the LWP package combines both web accessing and data extraction functions together, and thus can be an ideal tool for these purposes. Both cURL and LWP are freely available. cURL can be downloaded from `http://curl.haxx.se`, and for LWP, you can download Perl program language from `http://www.perl.com`, and LWP is a default package included in Perl.

Within SAS, SAS provides serveral Internet accessing tools, mainly with the FILENAME statement. The two main statements are FILENAME URL and FILENAME SOCKET statements. FILENAME URL is for basic web accessing with webpages using the GET request method. It can handle both HTTP and HTTPS, and also with webpages that need authentication (such as username/password). But FILENAME URL can not handle the POST request method. FILENAME SOCKET is for more complicate webpages, and can handle both GET and POST methods. To use FILENAME SOCKET statement, you need to understand the HTTP header. Helf (2005) provides some detailed explanation for the usage of this statement. Other FILENAME statements are FILENAME FTP and FILENAME EMAIL statements. As their names suggest, these two statements are for the FTP protocol and EMAIL accessing only.

Many web accessing can be handled by these SAS FILENAME statements. But in some situations, especially with webpages using the POST request method, it is difficult or too complicated to use these statements. We would like to use the cURL program or the LWP package. Fortunately, SAS provides two mechanisms to integrate external programs easily: the X statement and the FILENAME PIPE statement. The X statement lets you run a command line program (such as cURL) and then return back to SAS after the command line program finishes execution. You can use the *NOXWAIT* system option to get back to SAS automatically. The FILENAME PIPE statement lets you run the external program, feed the results from that program directly into SAS as an unamed pipe. You can think of this unamed pipe as reading from a local text file, except the content of this file is dynamically generated by the external program. With the X statement, if there is any return results that need to be processed by SAS, you need to save them in a local file, then use FILENAME statement to read this file. But for the FILENAME PIPE statement, there is no need to save the results in an intermediate file. The following examples illustrated how to use these two statement:

- Use cURL to obtain the home page of SAS and use the redirect sign (>) to save this homepage into a local file called *savedfile.txt*:

```
x "curl http://www.sas.com >savedfile.txt";
```

- With FILENAME pipe statement, you don't need to save the html file to a local file:

```
filename sashomepage pipe "curl http://www.sas.com";
```

For data extraction, SAS has many string functions and call rountines. We can use functions such as *FIND()*, *INDEX()*, *INDEXC()* to locate the position of the target information, and then use functions such as *SCAN()*, *SUBSTR()* to extract that information into a data set. Starting from Version 9, SAS provides a group of functions and call-rountines called Perl Regular Expression, all begin with the prefix *PRX*: *PRXCHANGE()* for replacement, *PRXMATCH()* for searching, *CALL PRXNEXT* for repeated extraction, etc. With the versatile Perl Regular Expression, these functions provide much flexible and powerful data extraction capabilities for SAS. Interested readers should check out their documentation in SAS.

EXAMPLES

EXAMPLE 1: DOWNLOAD .CSV FILE

Suppose you won't to download Moody's seasoned AAA coporate bond yield from the FRED (Federal Reserved Economical Data) website. The URL for this time series is `http://research.stlouisfed.org/fred2/series/DAAA/downloaddata/DAAA.csv`.

In this case, the webpage we are go to get is already organized in a .csv file, which can be read by SAS with an INFILE statement with the comma delimiter (dlm=","). If it is first downloaded in a local folder, say `C:\download`, and save it with the file name `DAAA.csv`, we can use the FILENAME statement to read it. The following SAS code shows how to get this time series from the local folder:

```
filename DAAA "C:\download\DAAA.csv";
```

But you don't need to download and save the file first, you can retrieve the .csv file directly from the server by using this code:

```
filename DAAA url "http://research.stlouisfed.org/fred2/series/DAAA/downloaddata/DAAA.csv"
```

EXAMPLE 2: OBTAIN THE LIST OF ESUG PRESENTATIONS

This example shows how to extract the target information from one single webpage. Suppose we want to get the list of all the archived presentations in the Edmonton SAS User Group (eSUG) meetings. We use the FILENAME URL statement to get the HTML file `http://www.sas.com/offices/NA/canada/en/edmonton.html`. And by examing the HTML file, we found that all the presentations are in the .pdf format, and the names are within the quotation marks, thus we look for the string within the quotation marks and ending with ".pdf". This can be done by first looking the lines that with the string ".pdf" (using FIND function), and then simply extract the second part of this line using the quotation mark (") as delimiter.

```
filename eSUG url "http://www.sas.com/offices/NA/canada/en/edmonton.html";
data eSUG_achive(keep=pdffile);
  length pdffile $200;
  infile eSUG length=len lrecl=32767;
  input line $varying32767. len;

  *all the file names end with .pdf;
  if find(line,".pdf") then do;
    *get the string ending with .pdf and enclosed by quotation marks;
    pdffile=scan(line,2,'"');
    output;
  end;

run;
filename eSUG clear;
```

What if we want to download all the pdf files. We could use FILENAME URL statement to obtain the pdf files, and then use the FILE statement to save them to an external file, but because the pdf file is a binary file, the FILE statement may not work. In this case, we use the X statement to let SAS run the cURL program with the –output option to download and save the whole file:

```
*concatenate the pdf links into a macro variable (PDFFiles);
proc sql noprint;
  select pdffile into :pdffiles separated by "|" from eSUG_achive;
quit;

options noxwait;
%macro downLoadFiles();
  %let eSUGFolder=H:\eSUG\PDFs;
  %let i=1;
  %let pdf1=%scan(&pdffiles.,&i.,"|");
  %do %while (%quote(&pdf1.)ËĒ=);
    *I only need the file name, don't need the path name;
    %let filename=%scan(&pdf1.,-1,"/");
    *the --output option saves the file;
    x "curl --output &eSUGFolder./&filename. &pdf1.";
    %let i=%eval(&i.+1);
```

```
    %let pdf1=%scan(&pdffiles.,&i.,"|");
  %end;
%mend downLoadFiles;
%downLoadFiles;
```

EXAMPLE 3: FIND OUT ALL CLINICAL TRIALS

Wei *et al* (2010) described how to use the FILENAME URL statement to obtain the registered clinical trials from the *www.clinicaltrial.gov* website. When the number of clinical trials meeting the search criteria is too large, the information of these trials is separated into multiple webpages. In Wei *et al* (2010), the authors first noticed that the page number is one of the parameters in the dynamic URL (e.g., pg=2), so they first obtain the total number of trials and number of trials per webpage, and then determine how many webpages need to be accessed. In this example, we use a more general web crawling method to fetch all the web page: if the link to “Next Page" is a legitimate URL (ie, begins with “href=", and follows with a string within the quotation marks), then we get this URL, put it in a macro variable “NextLink", otherwise this macro variable will be empty and the web crawling stops. This example also shows how to use the PRX-functions to locate and extract the target information in the HTML file.

```
%macro GetTrials(startlink=, out=Trials);

  %let Continue=Yes;
  %let NextLink=&startlink.;
  %let i=0;
  %do %while(&Continue.=Yes);

    filename Trials url "http://www.clinicaltrial.gov/%superq(NextLink)" lrecl=8192;

    data _Trials_(drop=recordline record nextpage);
      length Status $25 Study $200 link $200;
      retain Rank Status Study link;
      retain recordLine 0; *=1 means this line may contain needed information;
      infile Trials length=len;
      input record $varying8192. len;

      if prxmatch('/>Study<n/th>/i',record) then recordLine=1;
      if recordLine=1 and find(record,"/div") then recordLine=0;

      if (recordLine=1) then do; *now get the related information;
        if prxmatch('/<span.*>.*?<\/span>\s*$/',record) then
          status=prxchange('s/^.*?<span.*>(.*?)<\/span>\s*$/$1/',-1,record);
        if find(record,"href") then do;
          Study=prxchange('s/^.*>(.*?)<.*$/$1/',-1,record);
          link=prxchange('s/^.*?href="(.*?)".*$/$1/',-1,record);
        end;

        if find(record,"</table>") then do;
          rank=input(scan(link,-1,"="),5.0);
          output;
        end;
      end;

      if find(upcase(record),"NEXT PAGE") then do;
        if find(upcase(record),"HREF") then do;
          nextpage=htmldecode(prxchange('s/^.*?href="(.*?)".*?$/$1/i',-1,record));
          call symput("nextlink",nextpage);
        end;
        else call symput("Continue","No");
```

```
      end;
    run;
    filename Trials clear;

    data _Trials_;
      set _Trials_ end=last;
      if (last~=1) then output;
    run;

    proc append base=&out. data=_trials_;
    run;

    %let i=%eval(&i.+1);
  %end;
%mend GetTrials;
```

To use this macro, simply supply a start link (without the hostname of the website) and the name of the output data set. For example, to get all the clinical trials in Alberta, use the `ct2/results?state1=NA%3ACA%3AAB` as the start link (where `NA%3ACA%3AAB` is the encoded string of `NA:CA:AB`):

```
%GetTrials(startlink=%str(ct2/results?state1=NA%3ACA%3AAB),out=AB_Trials);
```

And for all trials in Canada:

```
%GetTrials(startlink=%str(ct2/results?state1=NA%3ACA),out=CA_Trials);
```

EXAMPLE 4: OBTAIN CITY OF EDMONTON'S JOB POSTINGS

In this example, we want to download all the job postings from the City of Edmonton Career website, extract all the relevant information and save them in a SAS data set for further analysis. The URL for the first webpage is *https://edmonton.taleo.net/careersection/2/moresearch.ftl*, which can be accessed by using the GET method. But if you want to see more pages or the details of each posting, you need to use the POST method. By examining the HTML codes of the first webpage, we found the "input" tags and the name and value attributes in each "input" tag. These are used for the construction of the query string to be sent in the content section of the POST method. By assigning a different value to some of the parameters, we can access other pages of postings and the details of each posting. Because the web accessing and the data extraction are quite complicated in this example, we use Perl/LWP to do most of the accessing and extraction, and in SAS we use the FILENAME PIPE statement to integrate the return results into a SAS data set. Since the Perl program has done most of the work, the SAS program becomes very simple. Below are the SAS codes and the Perl program is provided in the Appendix.

```
filename EdJobs pipe "C:\Perl\bin\perl C:\esug\edmjobs.pl";
data Edm_Jobs;

  *Totally 16 variables, 14 are read in as character strings. Declare their length;
  length JobCode $10 Title $100 Description $10000 Qualification $10000 Salary $500;
  length Hours $500 PDate $20 CDate $20 JobType1 $100 JobType2 $100 Union $100;
  length Department $100 WorkLocation $200 WorkAddress $200;

  infile EdJobs lrecl=32767 length=len;

  *use multiple input because the final result from Perl is printed in multiple lines;
  input JobCode $varying10. len;
  input Title $varying100. len;
  input JobNum;
  input Description $varying10000. len;
  input Qualification $varying10000. len;
  input Salary $varying500. len;
```

```
  input Hours $varying500. len;
  input PDate $varying20. len;
  input CDate $varying20. len;
  input NumOfOpening;
  input JobType1 $varying100. len;
  input JobType2 $varying100. len;
  input Union $varying100. len;
  input Department $varying100. len;
  input WorkLocation $varying200. len;
  input WorkAddress $varying200. len;
run;
filename EdJobs clear;
```

CONCLUSION

This paper shows how you can use SAS to access and extract information from the Internet, and demonstrates when the tools provided by the SAS system are not enough or too complicate to use, how you can integrate external programs (cURL and Perl) with the SAS system to help you get the work done. By writing SAS programs, or integrating other programs with the SAS system to automatically access and extract data from the Internet can help you increase the efficiency and accuracy of your work, and can update your SAS data sets automatically and timely when the Internet information is changed or updated.

REFERENCES

1. Garth Helf (2005), *Extreme Web Access: What to Do When FILENAME URL Is Not Enough*, Proceedings of the 30th SAS Users Group International Conference, http://www2.sas.com/proceedings/sugi30/100-30.pdf
2. Xin Wei, James Cai, Jim Rosinski (2010), *Use Base SAS URL to Build Surveillance and Monitoring System for New Clinical Trial Registration*, PharmaSUG 2010 Proceedings, http://www.pharmasug.org/cd/papers/AD/AD23.pdf

CONTACT INFORMATION

Comments and questions are welcomed and valued. Please contact the authors:

George Zhu (george.zhualbertahealthservices.ca)
Sunita Ghosh (sunita.ghosh@albertahealthservices.ca)
Cross Cancer Institute, Alberta Health Services - Cancer Care

ACKNOWLEDGEMENTS

A PERL PROGRAM *edmjobs.pl*

Here is the Perl/LWP program *edmjobs.pl*

```
#================================================================================================
# Perl program to download the job postings from City of Edmonton
# Written by George Zhu (george.zhu@albertahealthservices.ca)
# Usage:
#   1. Stand Alone:
#       C:\Perl\bin\perl C:\esug\edmjobs.pl >[output file name]
#   2. With SAS, use filename pipe statement:
#       filename EdJobs pipe "C:\Perl\bin\perl C:\esug\edmjobs.pl"
#      then in the data set step, use infile statement:
#       infile EdJobs lrecl=32767 length=len;
#     Note: change the folder name to where the program edmjobs.pl is located.
#================================================================================================
use strict;
use LWP::UserAgent;
use XML::Parser;
use URI::Escape;
use HTML::Entities;

my $EdJobs='https://edmonton.taleo.net/careersection/2/moresearch.ftl';
my $EdBrowser=LWP::UserAgent->new();
$EdBrowser->agent('Mozilla/5.0 (Windows NT 5.1; rv:6.0.2) Gecko/20100101 Firefox/6.0.2');

my $EdResponse=$EdBrowser->get($EdJobs);
my $EdPage1=$EdResponse->content() if $EdResponse->is_success();

## obtain the form items using the XML Parser;
my %inputAttrs; #for the POST data for next inquery;
my $EdParse=XML::Parser->new(Handlers=>\{Start=>\&Input_start,\});
sub Input_start
{
  my ($expat,$element,%attrs)=@_;
  if($element=~/input/i)  # look for the input tag
  {
    my ($param,$pmval);
    while(my($key,$value)=each(%attrs))
    {
      if ($key=~/name/i) {$param=$value;}
      if ($key=~/value/i) {$value=~tr/ /+/; $pmval=$value;}
    }
    $inputAttrs{$param}=$pmval;
  }
}

$EdParse->parse($EdPage1);
my %AttrsPage1=%inputAttrs;
my %AttrsPost1=%inputAttrs;

## total number of postings;
my $nJobs=$1 if $AttrsPage1{"initialHistory"}=~/listRequisition\.nbElements!\%7C!(\d+?)!\%7C!/i;
## Number of postings per page;
my $listSize=$1 if $AttrsPage1{"initialHistory"}=~/listRequisition\.size!\%7C!(\d+?)!\%7C!/i;
## calculate number of pages;
my $nPages=$nJobs/$listSize;
$nPages=int($nPages+1) if ($nPages>int($nPages));
```

```
## Obtain the pages of the postings, first set the parameters;
$AttrsPage1{"countryPanelErrorDrawer.state"}="false";
$AttrsPage1{"errorMessageDrawer.state"}="false";
$AttrsPage1{"ftlcompclass"}="PagerComponent";
$AttrsPage1{"ftlcallback"}="ftlPager_processResponse";
$AttrsPage1{"ftlcallback"}="ftlPager_processResponse";
$AttrsPage1{"ftlcompid"}="rlPager";
$AttrsPage1{"ftlinterfaceid"}="requisitionListInterface";

my $pp=1;
my $i=0;
my @JobCodes;
my $jobDetails='https://edmonton.taleo.net/careersection/2/jobdetail.ftl';
while ($pp<=$nPages)
{
  $AttrsPage1{"rlPager.currentPage"}="$pp";
  my $JobPage=$EdBrowser->post(jobDetails,\%AttrsPage1, 'Referer'=>EdJobs,);
  my @fillList=split("\',\'",$1) if $JobPage->content()=~/api\.fillList(.*?)\;/i;
  shift(@fillList);
  shift(@fillList);

  my $nLine=3;
  my $listSize=@fillList;
  while($nLine<$listSize)
  {
    $JobCodes[$i]=$fillList[$nLine];
    $nLine+=33;
    $i++;
  }
  $pp++;
}
$AttrsPost1{"ftlcompid"}="actOpenRequisitionDescription";
$AttrsPost1{"ftlinterfaceid"}="requisitionListInterface";

foreach my $code (@JobCodes)
{
  $AttrsPost1{"actOpenRequisitionDescription.requisitionNo"}="$code";
  # Use the POST method to get the detailed info about a posting
  my $Job1=$EdBrowser->post(jobDetails,\%AttrsPost1,'Referer'=>EdJobs,);

  $EdParse->parse($Job1->content());
  $inputAttrs{"initialHistory"}=~tr/+/ /;
  my @infor=split("!\%7C!",$inputAttrs{"initialHistory"});
  my $description=uri_unescape($infor[15]);
  decode_entities($description);
  $description=~s/(<.*?>)|(!\*!)//g;
  $description=~s/(\n)/ /g;

  my $qualification=uri_unescape($infor[16]);
  $qualification=~s/(<.*?>)|(!\*!)//g;
  decode_entities($qualification);

  my $salary=$1 if $qualification=~s/Salary Range: (.*?)\n//i;
  my $Hours=$1 if $qualification=~s/Hours of Work: (.*?)\n//i;
  $qualification=~s/(\n)/ /g;
```

```
  ## these are the extracted information from a job posting;
  print $code,"\n";          #jobcode
  print $infor[12],"\n";     #title
  print $infor[13],"\n";     #Job Number
  print $description,"\n";   #Description
  print $qualification,"\n"; #Qualification
  print $salary,"\n";        #Salary Range
  print $Hours,"\n";         #Hours of Work
  print $infor[20],"\n";     #Posting Date
  print $infor[22],"\n";     #Closing Date
  print $infor[23],"\n";     #Number of opening
  print $infor[25],"\n";     #Job type 1
  print $infor[26],"\n";     #Job type 2
  print $infor[27],"\n";     #Union
  print $infor[29],"\n";     #Department
  print $infor[32],"\n";     #Work location
  print $infor[33],"\n";     #Work location address
  #print "==================================\n\n";
}
####    End of program    ####
```