Paper A09-2009

An Example of Website "Screen Scraping"

Eric Lewerenz, My InnerView, Wausau, WI

ABSTRACT

Have you ever needed to collect information from a website without having to tediously cut-and-paste from several different web pages? This paper highlights a cobbled-together method the author used in solving a specific business problem. For beginner and intermediate SAS programmers, this paper may serve as an introduction to a wide range of different SAS functionality, including macros, regular expressions, the URL access method, the DO/%DO loop, PROC TRANSPOSE, and the INDEX and SUBSTR functions.

INTRODUCTION

Based on ongoing discussion at the Wikipedia website for the article on "Data scraping" (http://en.wikipedia.org/wiki/Data_scraping), there is some disagreement regarding the definition of "screen scraping." Wikipedia defines it (as of July 2009) as "a technique in which a computer program extracts data from human-readable output coming from another program." Putting lexical nuances aside, for practical purposes I believe "data scraping," "screen scraping" and "web scraping" would all be suitable terms for what I intended to do. The issue was that I wanted to extract specific and similarly-formatted information (name and address) from several web pages without having to laboriously cut-and-paste from each of them. Sounds like a job suited for SAS!

A SIMPLE REQUEST?

There was a management request to create a report for the National Association of State Veterans Homes (NASVH). Before this report could be created, we needed to know which customer facilities were also members of this association.

The NASVH has a website (www.nasvh.org) whereby you can look up member facilities by state. There is a page with a map, which has a dropdown of states.

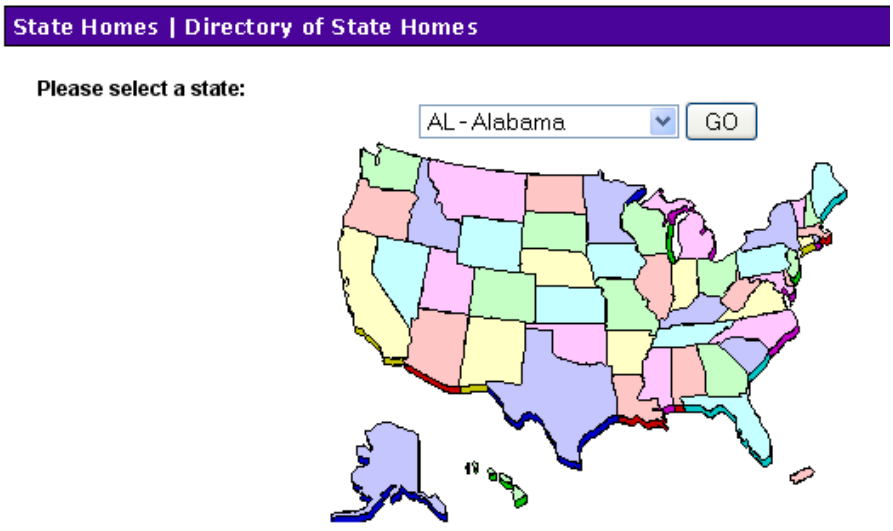

Figure 1. Screenshot from the NASVH website.

Choosing a state brings up a new page containing hyperlinked facility names – here is Minnesota.

Figure 2. Hyperlinked facility listing from NASVH website.

Clicking on a given hyperlink brings up a new page with a URL of the form:
http://www.nasvh.org/dir_statehomes/stateHome.cfm?ID=##

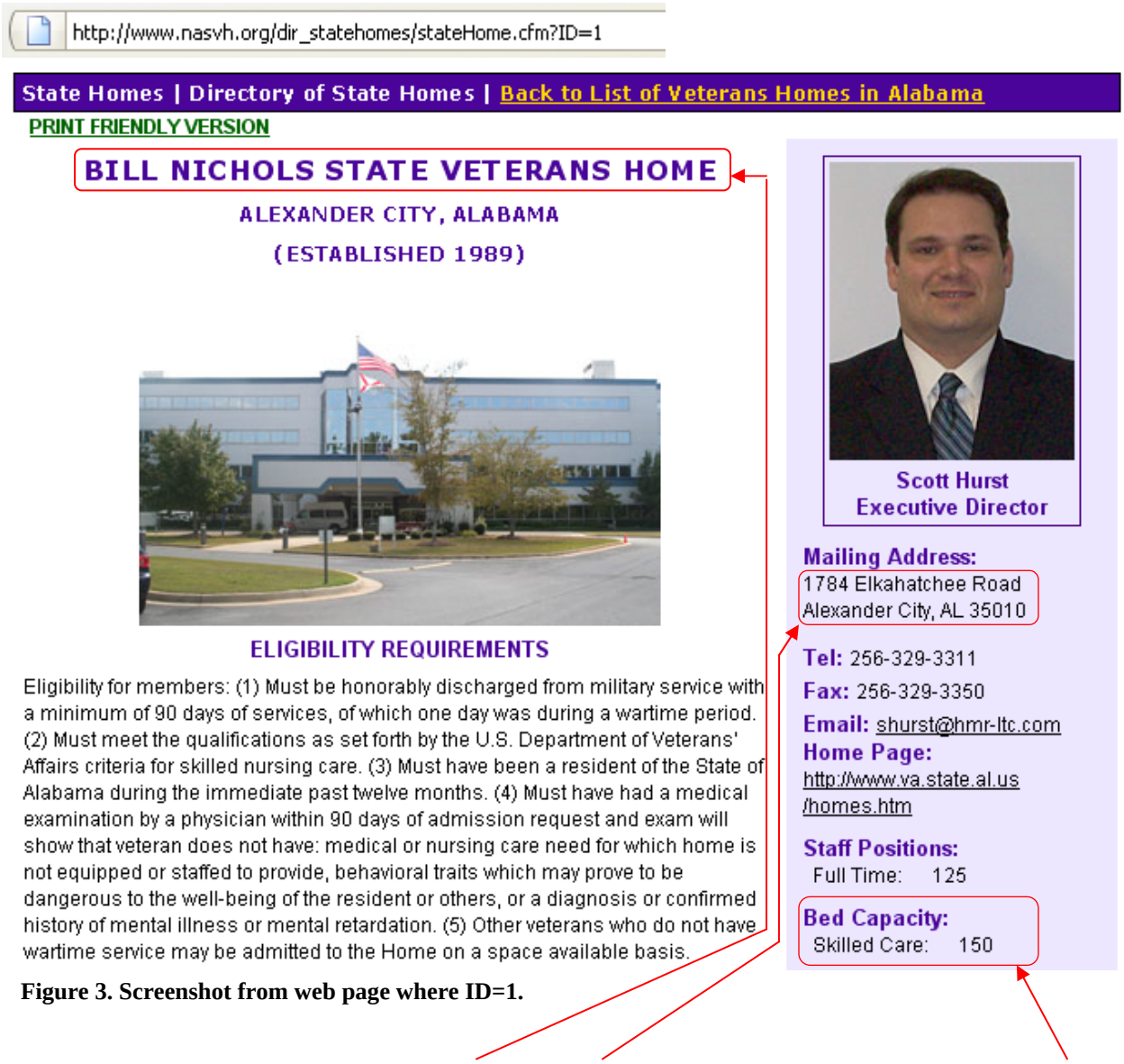

Figure 3. Screenshot from web page where ID=1.

What I needed to do was extract the facility's name and address. (I also attempted to extract the number of beds as well.)

PROBLEMS

There were several problems that had to be solved to get this to work, namely:

[1] How to read in data from a web page

[2] How to cycle through several web pages

[3] How to format the input

[4] How to parse the input and select specific records from similar attributes

SOLUTION (PART 1) – HOW TO READ IN DATA FROM A WEB PAGE

This was done using the URL access method. The basic code for this is:

```
FILENAME fileref URL 'external-file' <url-options>;
```

Here, *fileref* is the file reference name you assign, which will be used later when reading in the data; '*external-file*' is the URL for the web page; and <*url-options*> are different options you can invoke. In my program, the code looks like this:

```
FILENAME test URL "http://www.nasvh.org/dir_statehomes/stateHome.cfm?ID=1"
DEBUG LRECL=300;
```

I used the LRECL= option to set the logical record length of the input data. Using the 'DEBUG' option here causes SAS to write session information to the SAS LOG. In this case, the log looks like this:

```
NOTE: >>> GET /dir_statehomes/stateHome.cfm?ID=1 HTTP/1.0
NOTE: >>> Host: www.nasvh.org
NOTE: >>> Accept: */*.
NOTE: >>> Accept-Language: en
NOTE: >>> Accept-Charset: iso-8859-1,*,utf-8
NOTE: >>> User-Agent: SAS/URL
NOTE: >>>
NOTE: <<< HTTP/1.1 200 OK
NOTE: <<< Connection: close
NOTE: <<< Date: Mon, 30 Mar 2009 13:05:09 GMT
NOTE: <<< Server: Microsoft-IIS/6.0
NOTE: <<< X-Powered-By: ASP.NET
NOTE: <<< Set-Cookie: CFID=286621;expires=Wed, 23-Mar-2039 13:05:09 GMT;path=/
NOTE: <<< Set-Cookie: CFTOKEN=6fdf2bbc9d7cd4e2-577BC062-6094-3F32-29620C377E2B3DFF;expires=Wed,
      23-Mar-2039 13:05:09 GMT;path=/
NOTE: <<< Set-Cookie: CFID=286621;path=/
NOTE: <<< Set-Cookie: CFTOKEN=6fdf2bbc9d7cd4e2%2D577BC062%2D6094%2D3F32%2D29620C377E2B3DFF;path=/
NOTE: <<< Content-Language: en-US
NOTE: <<< Content-Type: text/html; charset=UTF-8
NOTE: <<<
NOTE: The infile TEST is:
      Filename=http://www.nasvh.org/dir_statehomes/stateHome.cfm?ID=1,
      Local Host Name=D6DWS5C1,
      Local Host IP addr=192.168.1.42,
      Service Hostname Name=www.nasvh.org,
      Service IP addr=208.194.177.83,
      Service Name=httpd,Service Portno=80,Lrecl=300,
      Recfm=Variable

NOTE: 308 records were read from the infile TEST.
      The minimum record length was 0.
      The maximum record length was 300.
      One or more lines were truncated.
NOTE: The data set WORK.TESTIN1 has 308 observations and 2 variables.
```

The rest of the basic code to create the data set looks like this:

```
DATA testin1;
id=1;
INFILE test length=len;
INPUT record $varying300. len;
```

```
RUN;
```

When you run this, it creates a data file ('testin1') with two variables: [i] id and [ii] record. 'Record' basically mimics what you would see if you instead looked at the page source via the web browser:

	id	record
50	1	
51	1	
52	1	
53	1	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xh
54	1	<html xmlns="http://www.w3.org/1999/xhtml"
55	1	<head>
56	1	▯<title>National Association of State Veterans Homes (NASVH) - Caring for America's Heroes</title>
57	1	▯<meta name="description" content="NASVH primary mission is to insure that every eligible veteran of the armed forces of America receive the benefits, services, and long term domiciliary care earned by their service and sacrifice.">
58	1	▯<meta name="keywords" content="veteran care, state home, veteran association, military, armed forces, military service, elderly, domiciliary care, military family services, patient care, residents, nursing programs, legislation, government, federal benefits, benefits, services">
59	1	▯
60	1	▯<link rel="stylesheet" type="text/css" href="../css/nasvh.css">
61	1	<link rel="stylesheet" type="text/css" href="../css/menu.css">
62	1	</head>
63	1	
64	1	<body>

Figure 4. Screenshot of contents of data set 'testin1.'

SOLUTION (PART 2) – HOW TO CYCLE THROUGH SEVERAL WEB PAGES

I noticed that the web page for each facility had a URL that ended in "?ID=##", so I reasoned that a macro could be built to cycle through a sequence of numbers to access each web page. After some trial-and-error, and reading from the NASVH website that there were only 137 member facilities, I decided that 150 would be a good upper-bound. (After reading in each of the web pages and counting the number of facilities, and noticing it was fewer than 137, I increased the upper-bound to 200, which got all of them.)

The macro took this form:

```
1     %MACRO ss(start,stop);
2     %DO numfacs = &start %TO &stop;
3
4     FILENAME test URL
5     "http://www.nasvh.org/dir_statehomes/stateHome.cfm?ID=&numfacs"
6     DEBUG LRECL=300;
7
8     DATA testin&numfacs;
9     id=&numfacs;
10    INFILE test length=len;
11    INPUT record $varying300. len;
12    RUN;
13
14    %END;
15    %MEND ss;
```

```
16
17      %ss(1,200);
```

What does all this mean? A macro was created called “ss” (which stands for screen-scrape), which takes in two parameters: *start* and *stop*. These two parameters provide the initial and ending values for the %DO loop. Line 2 of the macro creates a variable called “numfacs,” with initial value=*start* and ending value=*stop*. You can see in line 5 that “numfacs” now replaces the hard-coded ID value and is used to assign the “ID=” value for the website URL. This is how we loop through all the web pages. The end result of this step is that a data set is created for each web page (i.e., facility).

SOLUTION (PART 3) – HOW TO FORMAT THE INPUT

When I examined the contents of the “record” column, I found the rows that contained the data I needed. They looked like this (the data I needed is shaded in gray):

```
<tr><td class="h4" align="center">BILL NICHOLS STATE VETERANS HOME</td></tr>
<font class="content">1784 Elkahatchee Road<br>Alexander City, AL 35010     </font>
<br>  Skilled Care:      150
```

The next step was to write code that would keep only these rows. I started out using the INDEX function. With INDEX, you search for an instance of a character string. If the string is found, INDEX returns a non-zero number corresponding to the position of the first instance of the string. For example:

```
INDEX(record,'<font class="content">')
```

This will return a non-zero value for every row of the variable “record” in which it finds the character string `<font class="content">`; else it returns zero. The SAS code looks like this:

```
IF
(INDEX(record,'<td class="h4" align="center">') = 0) AND
(INDEX(record,'<font class="content">') = 0) AND
(INDEX(record,'Skilled') = 0) AND
(INDEX(record,'Domiciliary') = 0) THEN DELETE;
```

This reduced the number of rows of data kept, so now the created data file looks like this:

id	record
1	<tr><td class="h4" align="center">BILL NICHOLS STATE VETERANS HOME</td></tr>
1	1784 Elkahatchee Road Alexander City, AL 35010
1	Tel: 256-329-3311
1	 Fax: 256-329-3350
1	
1	
1	 Skilled Care: 150
1	Medical and nursing services, recreation therapy, physical therapy, occupational therapy, speech therapy, dietary services and church services.

Figure 5. Screenshot of contents of reduced data set ‘testin1.’

But you’ll notice that there are still some rows of data that are retained but unwanted; namely, telephone and fax (rows 3 and 4), some blank rows (rows 5 and 6), and some extraneous content (row 8). These all contain the character string “`<font class="content">`”. The next part was to exclude these rows, but still keep row 2 (address) by adding this:

```
ELSE IF (INDEX(record,'<font class="content">') NE 0) THEN DO;
```

```
        PATTERN=PRXPARSE("/A[LKSZRAEP]|C[AOT]|D[EC]|F[LM]|G[AU]|HI|I[ADLN]|K[SY]|
        LA|M[ADEHINOPST]|N[CDEHJMVY]|O[HKR]|P[ARW]|RI|S[CD]|T[NX]|UT|V[AIT]|W[AIV
        Y]/");
        IF (INDEX(record,'Tel:') NE 0) THEN DELETE;
        ELSE IF (INDEX(record,'Fax:') NE 0) THEN DELETE;
        ELSE IF (SUBSTR(record,33,3)='') THEN DELETE;
        ELSE IF PRXMATCH(PATTERN,record) = 0 THEN DELETE;
END;
```

Here, when the INDEX function finds an instance of the string ("…NE 0…"), it performs additional checks to see if it contains other character strings that we know we don't want ('Tel:', 'Fax:', or blank) or if the string does NOT contain one of the 2-character postal abbreviations for the 50 states (using a Perl regular expression), and deletes the row if one of these conditions is met.

PRXPARSE is used to define a Perl regular expression (which is just a pattern we want to find within a text string). I assign this pattern to a variable aptly called PATTERN. PRXMATCH makes use of PATTERN, and returns the position of PATTERN within the variable 'record' – a returned value of 0 means there was no match. Now the data set looks like this:

id	record	PATTERN
1	<tr><td class="h4" align="center">BILL NICHOLS STATE VETERANS HOME</td></tr>	.
1	1784 Elkahatchee Road Alexander City, AL 35010 	1
1	 Skilled Care: 150	.

Figure 6. Screenshot of contents of further reduced data set 'testin1.'

In the end, I'll need to combine all the data sets together and have all data specific to a given facility in the same row, not the same column. So, I simply transpose each newly-created data set within the same macro:

```
PROC TRANSPOSE DATA=testin&numfacs OUT=testout&numfacs;
       VAR record;
       BY id;
RUN;
```

A word on PROC TRANSPOSE: the variable that contains the data we want to transpose is listed in the VAR statement. The BY statement tells SAS that we want to transpose the data associated within each 'id,' so every time the value of 'id' changes would result in a new row.

And now the data set "testout1" looks like this:

id	NAME OF FORMER VARIABLE	COL1	COL2	COL3
1	record	<tr><td class="h4" align="center">BILL NICHOLS STATE VETERANS HOME</td></tr>	1784 Elkahatchee Road Alexander City, AL 35010 	 Skilled Care: 150

Figure 7. Screenshot of transposed data set 'testout1.'

SOLUTION (PART 4) – HOW TO PARSE THE INPUT AND SELECT SPECIFIC RECORDS FROM SIMILAR ATTRIBUTES

After combining all the "testout_" data sets, reviewing this combined data set reveals that the input code didn't always do what I wanted. (There was insufficient time to investigate the cause of this.) But for the most part, the complete address was either under COL2 or COL3. The following SAS code parses out, from the HTML, the facility name, as well as the address, city, and state from either of these columns:

```
DATA all3;
SET all2;
IF id=. THEN DELETE;
```

```
fn1=INDEX(COL1,'r">');
fn2=INDEX(COL1,'/td>');
fn=SUBSTR(COL1,fn1+3,fn2-fn1-4);

IF id NOT IN (10,17,31,80,109,110,167) THEN DO;
       ad1=INDEX(COL2,'t">');
       ad2=INDEX(COL2,'<br>');
       ad3=INDEX(COL2,',');
       addr=SUBSTR(COL2,ad1+3,ad2-ad1-3);
       city=SUBSTR(COL2,ad2+4,ad3-ad2-4);
       state=SUBSTR(COL2,ad3+2,2);
END;
ELSE IF id IN (10,17,31,80,109,110,167) THEN DO;
       ad1=INDEX(COL3,'t">');
       ad2=INDEX(COL3,'<br>');
       ad3=INDEX(COL3,',');
       addr=SUBSTR(COL3,ad1+3,ad2-ad1-3);
       city=SUBSTR(COL3,ad2+4,ad3-ad2-4);
       state=SUBSTR(COL3,ad3+2,2);
END;

IF id=104 THEN DELETE; /* Puerto Rico facility */
RUN;
```

After running this and then keeping only select variables, you get something like this:

id	fn	addr	city	state
1	BILL NICHOLS STATE VETERANS HOME	1784 Elkahatchee Road	Alexander City	AL
3	WILLIAM F. GREEN STATE VETERANS HOME	300 Faulkner Dr.	Bay Minette	AL
4	ALASKA STATE VETERANS AND PIONEERS HOME	250 East Fireweed	Palmer	AK
6	ARKANSAS STATE VETERANS HOME	4701 West Charles Bussey Avenue	Little Rock	AR
7	VETERANS HOME OF CALIFORNIA â€"BARSTOW	100 E. Veterans Pkwy.	Barstow	CA
8	VETERANS HOME OF CALIFORNIA - YOUNTVILLE	100 California Drive	Yountville	CA
9	VETERANS HOME OF CALIFORNIA - CHULA VISTA	700 East Naples Court	Chula Vista	CA
10	COLORADO STATE VETERANS CENTER	P.O. Box 97	Homelake	CO
11	COLORADO STATE VETERANS CENTER	P.O. Box 1420	Rifle	CO
12	BRUCE MCCANDLESS STATE VETERANS NURSING HOME	903 Moore Dr.	Florence	CO
13	COLORADO STATE VETERANS NURSING HOME	23500 U.S. Hwy. 160	Walsenburg	CO

Figure 8. Screenshot of finalized data set.

As you can see, the data are now in a much friendlier format to use! The final step was to export the data set to Excel, sort the records by state/city/address, and manually compare them against our customer database. If there had been more time, or the number of records had been much greater, it would probably have been more efficient to make use of some type of matching scheme, but time was a luxury in this case.

CONCLUSION

Often times when we code, we are trying to solve a pressing problem or to address an immediate need, and we don't always have the luxury of thinking about the best or most efficient way to write that code. We know what we know at the time, and can't afford to research other potential methods. I hope this paper has provided one clear method to screen-scrape using SAS. There are probably several other ways to use SAS to go about solving this problem. If you have developed your own methods, see ways to improve upon this code, or would like a copy of the complete program, please feel free to contact me at the email address below.

REFERENCES

Clay, Ted. 2006. "Tight Looping With Macro Arrays." *Proceedings of the Thirty-First Annual SAS User Group International Conference*, San Francisco, CA.

Cody, Ron. 2004. *SAS Functions by Example*. Cary, NC: SAS Institute Inc.

Tilanus, Erik. 2007. "Turning the data around: PROC TRANSPOSE and alternative approaches." *Proceedings of SAS Global Forum 2007*, Orlando, FL.

ACKNOWLEDGMENTS

I'd like to acknowledge my former supervisor, Brad Shiverick, for encouraging me to submit this paper. Never did I dream that something I did at work might bring value to my SAS peers.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Eric Lewerenz
eric.lewerenz@hotmail.com