Paper S1-22

Using SAS® to Build Web Pages Linking Photographs with Google Maps

Arthur S. Tabachneck, Ph.D., myQNA, Inc. Thornhill, Ontario Canada
William Klein, Ph.D., Toronto, Ontario Canada

ABSTRACT

Many of us at least occasionally take a picture with either a digital camera or cell phone. And some of us, be it for work, a SAS user group, or for personal use, end up posting some of our pictures on a web page. JPG files can contain a lot more information than just the pictures one sees, including such data as when and where pictures were taken, titles, picture heights and widths, and many of our camera's settings. This paper includes all of the base SAS code needed to create web pages that show one's pictures, in date taken order, with links that, when clicked, will cause Google Earth to show you to where the pictures were taken.

OVERVIEW

Accomplishing the task really isn't very difficult if you either just run the code included with this paper or, on the other hand, are familiar with:

- the use of pipes to run a system command to locate all of a directory's relevant files
- the exif file specifications that are used for virtually all JPG files
- writing code to read hexadecimal characters, including numbers in both big and little endian form
- using functions to find the location of particular strings within a file
- creating html files without the aid of any tool other than base SAS
- using the Google Map application program interface (or *API* as it is commonly known)

PURPOSE

The paper has two purposes. One purpose is to provide code that will enable you to build web pages, like the one described earlier, without having to know why or how the method works. The other purpose, if you might be interested in knowing why the method works and possibly improve on it or generalize it to other applications, is to provide overviews of the six concepts and explanations of how each was applied to accomplish the present task.

Satisfying the first purpose is easy. Copy and paste the code (Appendix 1) into your SAS editor and run the code. Of course, you may first want to modify the eight macro variables at the beginning of the code to specify where your jpg files are located, the name of the desired output file, the title and subtitle you want to assign to the web page, how you want null titles displayed, whether you want Global Positioning System (GPS) coordinates displayed, and whether you want your pictures to be numbered. And, if you aren't on a windows-based system, you may also have to change the ninth line to reflect your system's directory command. Then, simply run the code.

Regarding the second goal, we are definitely not suggesting that one use SAS to uncover all of the metadata that is captured within JPG files (although you probably could). The metadata can easily be extracted with a language like PERL, but developing such code isn't necessary, as others have already done so and made the code available for free in forms that don't even require one to have PERL installed on their computer (see, e.g., http://www.sno.phy.queensu.ca/~phil/exiftool/).

However, many of us work in environments where downloading freeware isn't permitted, yet we may want (or need) to accomplish a task similar to the one performed by the code provided in this paper.

DETAILS

Lines of code you may have to change. The present code uses eight macro variables to enable users to quickly identify and change eight of the nine things that may have to be changed in order to run the code presented in this paper. We used macro variables for these factors because their application occurs later in the code and we wanted to ensure that users only have to modify the first nine lines of the code. The eight macro variables are created with the following statements:

1. To specify the root path where your jpg files are located:
```
%let path=c:\;
```
2. To indicate the filename you want for your output file:
```
%let outfile=my_pictures.html;
```
3. To specify the caption, if any, that you want to precede your pictures' titles:
```
%let caption=;
```
4. To indicate the title you want shown at the top of the web page:
```
%let webpagetitle=Our Vegas/Grand Canyon Trip;
```
5. To indicate a subtitle or message you want shown under the web page title:
```
%let webpagenote=Note: If your mouse pointer appears as a hand when
 you move it over a picture, left-click your mouse to have Google
 Earth show the location of where the picture was taken;
```
6. To indicate whether you want a sequential number shown to the right of your pictures' titles:
```
%let afterpicturenumber=no;
```
7. To specify the string to be used for pictures that don't have a title. If left blank the file name will be used:
```
%let defaultpicturetitle=;
```
8. To specify whether GPS links, if available, should be included:
```
%let includeGPSlinks=YES;
```

Using the pipe device type to locate all relevant files from a directory. Using the pipe device type, combined with the SAS filename statement and your operating system's directory command, provides an extremely convenient way to create a file that contains a set of records, one record for each file name, with each record containing the name of one of the files. The method is applicable for all operating systems, although the code provided in the present paper was written to run on Windows operating systems, and thus uses the pipe device type to run the DOS *dir* command. The specific code used in the program was:

```
filename indata pipe "dir &path.*.jp* /b";
```

Since the above code uses the */b* option for the DOS *dir* command, the code will cause SAS to create a file, called *indata,* that will contain the names of every picture (i.e., any file with an extension that begins with the string "jp") that exists in the root directory *c*:\. While other options could also be used, the code was written to expect filenames that aren't preceded by the files' paths. That is accomplished by using the */b* option.

Exif File Structure. In developing the present code we needed discover how one could use SAS to read jpg files' metadata which, if present, follows a standard structure known as the *Exchangeable Image File Format* or *EXIF*. An understandable explanation of the structure can be found at: http://gvsoft.homedns.org/exif/exif-explanation.html and the actual specifications can be found at: http://www.exif.org/Exif2-2.PDF

JPG files aren't comprised of rows and columns but, rather are simply a set of characters with some of the characters representing tags that define the length and format of the characters that follow them. And, while the file sizes can be fairly large, the information the current program was designed to extract, if present, is always located within each file's first 15,000 characters. The code was written to extract as much of the desired metadata as possible and, if any of the needed information can't be found, to attempt alternative methods for determining such things as picture height and width, desired picture orientation and the dates on which pictures had been taken.

The first thing the code does is search for the string *Exif*. We used the index function to accomplish the task and, if the program doesn't find the string in a particular file, an alternative process is initiated.

If the string Exif is found, the code relies on the file conforming to the *Exif* file structure, in order to parse the information it is expected to contain. Six characters following the location of the string *Exif are* two characters that represent a code which signifies whether the numbers throughout the rest of the file are in big or little endian format. We'll talk more about that in the next section but, basically, endianness is simply whether numbers should be read from left to right (big endian), or from right to left (little endian). The Wikipedia article on *Endianness* provides quite a bit of background on the topic (see: http://en.wikipedia.org/wiki/Endianness).

The code and all of the numeric information throughout the file are defined in the hexadecimal representation of the numbers. The position of the endian code indicator in the file is also considered position 0 and all locations specified in the rest of the file are stated in relative terms using that position as the starting point. In relative positions 3-6 there is a four character hexadecimal number that, when read in the correct endian order, provides the location of a two character hexadecimal number that indicates the number of 12-character *Exif IFD0* tags that immediately follow it. The tags are described in the first 2 characters of each block of twelve characters and the other 10 characters have different, but standard meanings for each tag. While there are typically about 13 or more such tags, we were only interested in three of them:

> *xtag 8769x* (or 34665) describes where the counter for *the Exif IFD* tag values will be found. Note this and all tags are dependent upon endian order. Thus, if big endian is used, the tag will be 8769x but, for little endian, the tag would be 6987x.
>
> *xtag 8825x* (or 34853) describes where the latitude and longitude coordinates can be found which indicate where the picture was taken.
>
> *xtag 9c9bx* (or 40091) indicates where a picture's title can be found, as well as the title's length.

The dates the pictures were taken, as well as the pictures' heights and widths, are found in the IFD tags:

> *xtag 9003x* (or 36867) describes where one will find the datetime that the picture was taken.
>
> *xtag a002x* (or 40962) contains the picture's width.
>
> *xtag a003x* (or 40963) contains the picture's height.

A tool we found to be invaluable in deciphering the *Exif* file structure was the program called *exiftool* mentioned earlier (i.e., http://www.sno.phy.queensu.ca/~phil/exiftool/). If one runs that program with the following command: exiftool –htmldump *filename*.jpg > *filename*.html the program will produce an html file that makes it very easy to see where all of the fields are located and how they are structured.

Another tool we found to be extremely helpful was the on-line hex to decimal converter that can be found at: http://easycalculation.com/hex-converter.php.

Correcting for Endianness. Since a jpg file's *edianness* is captured in a two character field that is located six characters past the start of the string *Exif*, we were able to capture the value with the following code:

```
positionx = index(var1,'Exif');
if positionx gt 0 then positionx+6;
```

Thus, *positionx* was established so that we could both discover each file's *endianness* and know the position which all of the *Exif tags* would use as a reference point.

The possible codes used to represent endianness in an *Exif* section are "4949"x (i.e., 18761 or 'II') for little endian, and "4D4D"s (i.e., 19789 or 'MM') for big endian. We addressed the endianness differences by creating a format as follows:

```
proc format;
  value tendian 18761='pibr.'
                19789='s370fpib.';
run;
```

We then applied the format to the endian code that was extracted from the file's relative starting position:

```
endian=put(input(substr(var1,positionx,2),pibr2.),tendian.);
```

Using the above approach allowed us to use the *inputn* function to extract all numeric data as we could apply *endian* as a format and specify the width based upon the values indicated by the various Exif IFD0 and IFD tag values. For example, to capture each picture's width, we only had to include the following code:

```
if xtag eq 40962 then width=
 inputn(substr(var1,last_offset+i*12+8,4),endian,4);
```

Creating an HTML File. H*ypertext Markup Language* (i.e., *HTML*) is the predominant method used today for creating web pages. The files are simply text files that follow the format specified by the *World Wide Web Consortium or WS3*. The details and available options, tags and attributes one can use to create such files are well beyond the scope of the present paper and are readily available from a number of sources. A nice introduction, including numerous references, can be found in the *HTML* Wikipedia article (see: http://en.wikipedia.org/wiki/HTML).

But, one doesn't even have to learn the markup language in order to be able to create web pages as there are numerous commercial and free on-line *WYSIWYG (What You See Is What You Get*) html editors available (see, e.g., http://en.wikipedia.org/wiki/HTML_editor). And, since the code they produce are simple text files, the code can easily be copied and pasted into SAS code which only has to be quoted and prefaced with put statements in order to develop a program like the one presented in the present paper.

However, taking on such a task isn't even necessary as the code, included as Appendix 1 of this paper, already creates an html file that can be used as is. Of course the code could be modified if desired.

The Google Map Application Program Interface. While we felt that the previously mentioned code already provided a sufficient amount of new techniques for SAS users, we decided to include one additional feature namely giving users the capability of being able to click on any picture and automatically bring up a map showing where each picture was taken. In all honesty, we were skeptical about adding such a feature for fear that the code would become overly complex.

To our pleasant surprise, only a couple of very simple lines of code were needed. The *Google Map Application Program Interface* (i.e., *API*) is now a web address that will accept map coordinates in a hyperlink.

The code that had to be added was:

```
if coordinates eq "No GPS" or upcase("&includeGPSlinks.") eq "NO" then
  put '<img src=' picture ' width="' width '" height="' height '"></td></tr><tr>';
else put '<a href="http://maps.google.com/maps?f='
         'q&source=s_q&hl=en&geocode=&q=' coordinates
         '&aq=&ie=UTF8&t=f&z=16&vpsrc=0&ecpose=' coordinates
         ',1025.22,0,0,0"> <img src=' picture ' width="' width
         '" height="' height '"></a></td></tr><tr>';
```

The last five lines in the code shown above code output the link that is necessary to access Google Earth. The word *coordinates*, which appears twice in those lines, represents the values of the field *coordinates* that was captured in the previous datastep. Similarly, the words *width* and *height* in the above lines of code, represent the picture widths and heights that were extracted from each file and adjusted to fit in a standard space. Of course, the link will only work if you are connected to the internet.

An explanation of what the code does is most easily accomplished by showing an example of the page that will appear in one's browser after the code has been run and one double clicks on the file that is created.

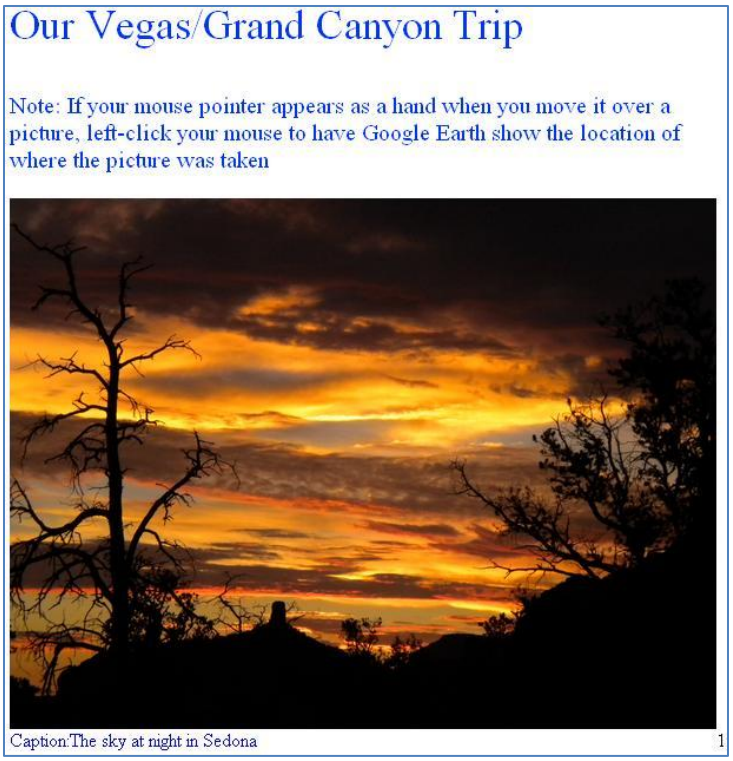

Figure 1 – Example result of running the program's code

If you then click on any picture that includes GPS coordinates, a screen like the one shown, below, will appear:

Figure 2 – Example Google Map resulting from running the program's code

And, not only do you get to see where the picture was taken, but you can manipulate all of the ActiveX controls, thus you can zoom in or out, change the altitude, and move any distance in any direction. Thus, if you didn't know where Sedona, Arizona was, you could easily zoom out until you were far enough away, like the screen shot shown below, where you could easily see that Sedona, AZ is just south of the Grand Canyon and South-East of Las Vegas.

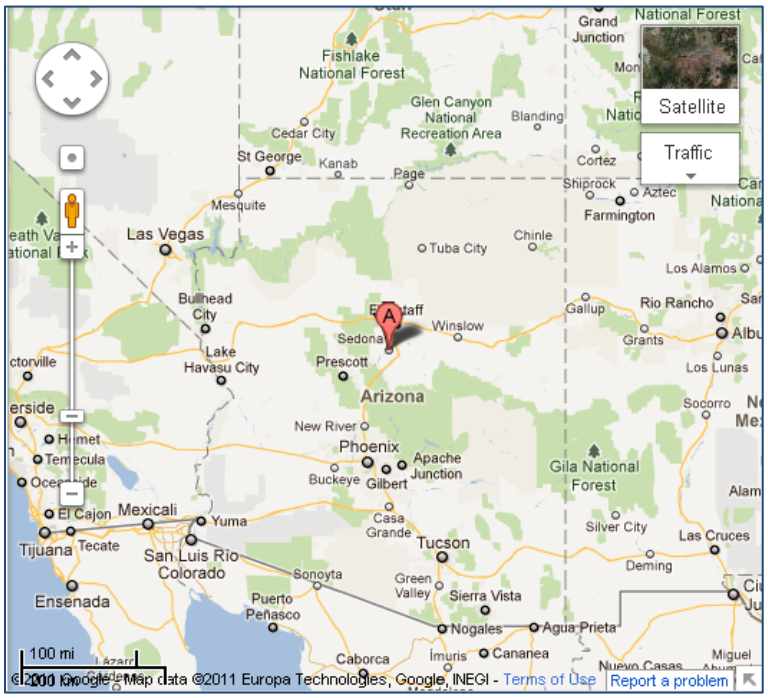

Figure 3 – Example Google Map resulting from running the program's code

Adding metadata to your jpg files. One can use Window's explorer, or any of the many applications that are available, to change or add to a JPG file's metadata. We strongly recommend that you first backup your files before attempting to add things like titles and GPS coordinates, as many programs can corrupt the existing *Exif* file structure. Since the *Exif IFD* tags identify such information as when pictures were taken, where, and picture height and width, such a disruption could be quite detrimental to how the present code uses that data.

A very good tool for making such changes, and reviewing all of your files' metadata, is the freeware *exiftool* mentioned earlier in this paper (i.e., http://www.sno.phy.queensu.ca/~phil/exiftool/).

With *exiftool* adding a title to a JPG file only requires one command prompt line:

```
exiftool -xptitle="Whatever title you want" yourfilename.jpg
```

Similarly, if your camera isn't GPS capable and you wanted to add coordinates to a picture, you could do so with statements like those shown in the following two lines:

```
exiftool -exif:gpslatitude="28 21 59.53" -exif:gpslatituderef=N a.jpg
exiftool -exif:gpslongitude="81 33 38.32" -exif:gpslongituderef=W a.jpg
```

The two statements, above, contain the degrees, minutes, seconds and references for both a coordinate's latitude and longitude. The coordinates shown in the two statements happen to be the coordinates of the Walt Disney World Swan and Dolphin Resort.

Conversely, while the Windows Explorer properties' menu doesn't allow one to add GPS coordinates, it can definitely be used to add or change such values as a file's height, width and title. In fact, we found it to be quite convenient in adding picture titles that were written using some French characters.

SOME FINAL NOTES AND FUTURE CHANGES TO THE CODE

While we tried to ensure that tag identifiers and locations were used correctly in the code presented in this paper, and the code successfully passed all of our tests, there could still be errors and omissions. For example, sometimes jpg files don't contain an *Exif* section, and sometimes the files contain an *Exif* section but the section doesn't include picture heights and widths. Similarly, sometimes pictures in jpg files have been rotated (e.g., from landscape to portrait representation), but the file's *Exif* metadata continue to reflect the dimensions prior to the rotation.

As such, if picture height and width aren't found, the code performs additional checks in an attempt to parse those values from another metadata component that might be present in the files, namely the *JPEG File Interchange Format* or *JFIF* section. If a picture's height and width aren't found in the Exif section, the code attempts to find that data in the *JFIF* section.

One check made in the code is based on a commonality we discovered in the files we tested but, for which, we were unable to find supporting documentation. Specifically, if a picture's height and width are found in the *Exif* section, and picture width is greater than picture height but the file's fourth character contains the value 'E0'x, it is presumed that the picture has been rotated such that the dimensions are reversed from those shown in the file.

We found that the above mentioned checks for orientation resulted in correctly displaying all of the files we tested. However, we included an override condition so that users will have a way to force any picture to be shown as tall and narrow (i.e., portrait orientation, where a picture's height is greater than its width). To ensure that a picture will be displayed as being tall and narrow, add the string "VERT" to the end of its name (e.g., myfileVERT.jpg).

Although the code was carefully written and tested, the code may have to be updated to correct for errors that we or others might discover. As such, we've created a page for the paper on sasCommunity.org. The page includes the powerpoint presentation and a machine readable copy of the code, and we will update the files on that page as necessary. The page can be found at:
http://www.sascommunity.org/wiki/Using_SAS_to_Build_Web_Pages_Containing_Ones_Pictures.

DISCLAIMER

The contents of this paper are the work of the authors and do not necessarily represent the opinions, practices, or recommendations of their respective organizations.

CONTACT INFORMATION

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

Arthur Tabachneck, Ph.D.
myQNA, Inc..
Thornhill, ON Canada
E-mail: atabachneck@gmail.com

William Klein, Ph.D.
Toronto, Ontario Canada
E-mail: william.klein@utoronto.ca

APPENDIX I

```
/*********************************************************/
/*Program:   get_pic_info.sas                            */
/*Purpose:   To create a web page based on metadata      */
/*           extracted from a set of jpg files           */
/*Created:   27SEP2011                                   */
/*Modified:  23DEC2011 02JUN2012 16JUL2012 17JUL2012     */
/*Author(s): Bill Klein and Arthur Tabachneck            */
/*********************************************************/

/*Modify the following nine lines, if necessary, to      */
/*indicate:                                              */

  /*1 the path where your jpg files are located          */
  %let path=c:\art\;

  /*2 the name you want for your output file             */
  %let outfile=my_pictures.html;

  /*3 the caption, if any, that you want placed before   */
  /*  all picture titles                                 */
  %let caption=Caption: ;

  /*4 the title you want shown at the top of the web page*/
  %let webpagetitle=Our Vegas/Grand Canyon Trip;

  /*5 the note, if any, you want shown immediately below */
  /*  the web page title                                 */
  %let webpagenote=Note: If your mouse pointer appears
    as a hand when you move it over a picture,
    left-click your mouse to have Google Earth show the
    location of where the picture was taken;

  /*6 whether you want a number shown after your pictures'*/
  /*  titles                                             */
  %let afterpicturenumber=yes;

  /*7 the string, if any, that you want to use for       */
  /*  pictures that don't have a title.  If left blank   */
  /*  the file name will be used                         */
  %let defaultpicturetitle=;

  /*8 whether you want GPS links added to the pictures   */
  %let includeGPSlinks=YES;

  /*9 your operating system's directory command. This    */
  /*  statement will ONLY have to be changed if you are  */
  /*  NOT using a MS Windows-based system.               */
  /*                                                     */
  /*   The statement assigns the filename "indata" using */
  /*   the pipe access method to run your operating      */
  /*   system's directory command to identify all jpg    */
  /*   files in &path.                                   */
  filename indata pipe "dir &path.*.jpg /b";

/*Set the datestyle system option so that the anydtdtm   */
/*informat will expect datetimes to show year:month:day  */
options datestyle=ymd;

/*Create a format to differentiate between big and little*/
/*endian numbers based on the Exif IFD0 endian code      */
proc format;
  value tendian 18761='pibr.'
                19789='s370fpib.';
```

```
run;

/*The following datastep reads all of the identified jpg   */
/*files and, if a file contains Exif metadata, attempts    */
/*to extract picture titles, dimensions, dates pictures    */
/*were taken, and GPS coordinates                          */
data want (keep=pi: dt: c: title w: he:);
  length fil2read title $420;
  format dt_taken datetime19.;
  format lat lon $1.;
  length coordinates $35;
  infile indata truncover;

  /* Read the filenames identified by your system's        */
  /* directory command                                     */
  informat picture $100.;
  input picture &;

  /* The following statement concatenates &path with each  */
  /* filename                                              */
  fil2read="&path."||picture;

  /* The following lines read the first 15,000 characters  */
  /* of each jpg file                                      */
  done=0;
  infile dummy filevar=fil2read RECFM=n lrecl=15000 end=done;
  if not done then do;
    input VAR1 $char15000.;
    link initialize;

    /* If an Exif header is found, the following lines     */
    /* attempt to extract the desired Exif metadata        */
    if positionx gt 0 then do;
      link parse_Exif;
      link parse_ifd0;
      link parse_ifd;
      link get_title;
      link get_gps;
      link get_date;
    end; /* end of exif specific checks                    */

    link get_jfif;
    link final_adjust;

    /* Output record and set done to 1 to get next jpg file*/
    output;
    link lastline;

    initialize:
         /* Initialize variables                          */
      coordinates="No GPS";
      if length("&defaultpicturetitle") lt 2 then
       title=picture;
      else title="&defaultpicturetitle";
      pheight=0;
      pwidth=0;
      jheight=0;
      jwidth=0;
      portrait=0;
      dt_taken=0;

      /* Initialize all relevant Exif IFD offsets          */
      gps_offset=0;
      subject_offset=0;
      date_offset=0;
      last_offset=0;
```

```
  /*The following line finds identifies whether an Exif*/
  /*header exists in the file and, if so, the position */
  /*that serves both as the location of the Exif endian*/
  /*code identifier and Exif relative position #1      */
  positionx = index(var1,'Exif');
return;

parse_Exif:
  positionx+6;
  /* The following line creates the variable endian    */
  /* which is used to specify the correct format to    */
  /* use for reading all numbers                       */
  endian=put(input(substr(var1,positionx,2),
             pibr2.),tendian.);

  /* Read the value that specifies the number of Exif  */
  /* IFD0 tags that have to be read                    */
  numberx=inputn(substr(var1,positionx+inputn(substr(
   var1,positionx+4,4),endian,4),2),endian,2);

  /* Identify the offset of the first IFD0 tag         */
  offset=positionx+inputn(substr(var1,positionx+4,4),
   endian,4)+2;
return;

parse_ifd0:   do i=0 to numberx-1;
  select (inputn(substr(var1,offset+i*12,2),endian,2));
    when (34665) do;
      numberx2=inputn(substr(var1,positionx+inputn(substr(var1,
       offset+i*12+8,4),endian,4),2),endian,2);
      last_offset=positionx+inputn(substr(var1,offset+i*12+8,4),endian,4)+2;
    end;
    when (34853) do;
      gps_offset=positionx;
      if inputn(substr(var1,offset+i*12+2,2),endian,2) eq 2
       then gps_offset+inputn(substr(var1,offset+i*12+10,2),endian,2);
      else gps_offset+inputn(substr(var1,offset+i*12+8,4),endian,4);
    end;
    when (40091) do;
      title_offset=positionx+inputn(substr(var1,offset+i*12+8,4),endian,4);
      title_length=inputn(substr(var1,offset+i*12+4,4),endian,4);
    end;
    otherwise;
  end;
   end;
return;

parse_ifd:  if last_offset then do i=0 to numberx2-1;
  select (inputn(substr(var1,last_offset+i*12,2),endian,2));
    when (36867) date_offset=inputn(substr(var1,last_offset+i*12+8,4),endian,4);
    when (40962) do;
      if inputn(substr(var1,last_offset+i*12+2,1),endian,1) eq 4 then len=4;
      else len=2;
      pwidth=inputn(substr(var1,last_offset+i*12+8,len),endian,len);
    end;
    when (40963) do;
      if inputn(substr(var1,last_offset+i*12+2,1),endian,1) eq 4 then len=4;
      else len=2;
      pheight=inputn(substr(var1,last_offset+i*12+8,len),endian,len);
    end;
    otherwise;
  end;
end;
return;
```

```
get_title:
  /* Based on whether title and GPS data are found, set*/
  /* the title and GPS variables accordingly           */
  if 1<=title_offset<=15000-title_length then
   title=compress(inputc(substr(
   var1,title_offset,title_length),title_length),,"c");
return;

get_gps:  if 1<=gps_offset<=15000-24 then do;
  numberx=inputn(substr(var1,gps_offset,2),endian,2);
  offset=gps_offset+2;
  do i=0 to numberx-1;
    xtag=inputn(substr(var1,offset+i*12,2),endian,2);
    if xtag eq 1 then lat=input(substr(var1,offset+i*12+8,1),$1.);
    else if xtag eq 2 then do;
      lat_offset=positionx+inputn(substr(var1,offset+i*12+8,2),endian,4);
      latdeg=inputn(substr(var1,lat_offset+ 0,4),endian,4)/
       inputn(substr(var1,lat_offset+ 4,4),endian,4);
      latmin=inputn(substr(var1,lat_offset+ 8,4),endian,4)/
       inputn(substr(var1,lat_offset+12,4),endian,4);
      latsec=inputn(substr(var1,lat_offset+16,4),endian,4)/
       inputn(substr(var1,lat_offset+20,4),endian,4);
    end;
    else if xtag eq 3 then lon=input(substr(var1,offset+i*12+8,1),$1.);
    else if xtag eq 4 then do;
      lon_offset=positionx+inputn(substr(var1,offset+i*12+8,2),endian,2);
      londeg=inputn(substr(var1,lon_offset+ 0,4),endian,4)/
       inputn(substr(var1,lon_offset+ 4,4),endian,4);
      lonmin=inputn(substr(var1,lon_offset+ 8,4),endian,4)/
       inputn(substr(var1,lon_offset+12,4),endian,4);
      lonsec=inputn(substr(var1,lon_offset+16,4),endian,4)/
       inputn(substr(var1,lon_offset+20,4),endian,4);
    end;
  end;
  decimal_lat=ifc(lat eq "N",latdeg+(latmin*60+latsec)/
   3600,-1*(latdeg+(latmin*60+latsec)/3600));
  decimal_lon=ifc(lat eq "E",londeg+(lonmin*60+lonsec)/
   3600,-1*(londeg+(lonmin*60+lonsec)/3600));
  if nmiss(of dec:) lt 1 then coordinates=
   catx(",",decimal_lat,decimal_lon);
end;
return;

get_date:
  /* If found, parse date picture was taken            */
  if 1<=date_offset<=15000-19 then dt_taken=
   input(substr(var1,positionx+date_offset,19),
    anydtdtm19.);
return;

get_jfif:
  /* search for a JFIF header and, if found, attempt to*/
  /* find the JFIF height and width                    */
  if index(var1,'JFIF') then do;
    positionx = index(var1,'FFC0'x);
    if positionx gt 0 then do;
      jheight=input(substr(var1,positionx+5,2),s370fpib2.);
      jwidth=input(substr(var1,positionx+7,2),s370fpib2.);
      if jheight gt 0 and pheight le 0 then do;
        pheight=jheight;
        pwidth=jwidth;
      end;
    end;
  end;
  /*if necessary, assign default height and width      */
  if pheight eq 0 then do;
```

```
        pheight=450;
        pwidth=600;
      end;
    return;

    final_adjust:
      /*If filename ends with the string "vert" ensure that */
      /*the picture is displayed with portrait rather than  */
      /*landscape dimensions                                */
      if length(reverse(scan(picture,-2,"."))) gt 4 then
       if upcase(substr(reverse(scan(picture,-2,".")),1,4))
        eq "TREV" then portrait=1;

      /* Add title caption                                  */
      title="&caption"||title;

      /* If the portrait flag was identified, and width     */
      /* is greater the height, then switch the             */
      /* values of pheight and pwidth                       */
      if portrait and pwidth gt pheight then do;
        holdw=pwidth;
        pwidth=pheight;
        pheight=holdw;
      end;

      /* Resize Pictures                                    */
      if pheight gt pwidth then do;
        height=pheight+(pheight*(600/pheight-1));
        width=600*(1+(pwidth/pheight-1));
      end;
      else if pwidth gt pheight then do;
        height=600*(1+(pheight/pwidth-1));
        width=pwidth+(pwidth*(600/pwidth-1));
      end;
      else do;
        height=600;
        width=600;
      end;

      if dt_taken eq 0 then do;
      /* If no date was found attempt to find one using a   */
      /* regular expression pattern search                  */
        _dt_pattern_num=prxparse(
         "/\d\d\d\d\:\d\d\:\d\d\ \d\d\:\d\d\:\d\d/o");
        date_offset = prxmatch(_dt_pattern_num,var1);
        /* If found, then parse date                        */
        if date_offset then dt_taken=
         input(substr(var1,date_offset,19),anydtdtm19.);
      end;
    return;

    lastline:  end;
run;

/* Sort the file according to the dates that pictures      */
/* were taken                                              */
proc sort data=want;
  by dt_taken;
run;

/* Create HTML file                                        */
data _null_;
  file "&path.&outfile";
  set want end=done;
  if _n_=1 then put  /* Start with header information      */
   '<html> '
```

```
   / '<head> '
   / "<title>&webpagetitle</title>  "
   / '<meta http-equiv="Content-Type' '" content='
     '"text/html; charset=iso-8859-1">'
   / '<style type="'  'text/css">'
   / '<!--                    '
   / '.style4 {               '
   / '     font-size: 36px;        '
   / '     color: #0033FF;         '
   / '}                            '
   / '-->                          '
   / '</style>                     '
   / '<style type="'  'text/css">'
   / '<!--                    '
   / '.style2 {               '
   / '     font-size: 20px;        '
   / '     color: #0033FF;         '
   / '}                            '
   / '-->                          '
   / '</style>                     '
   / '</head>                      '
   / '<body>'
   / '<p class="style4">'"&webpagetitle</p>"
   / '<p class="style2">'"&webpagenote</p>"
   /
   / '<table width="600" border="0"'
     'cellpadding="0" cellspacing="0">';

  put /'   <tr><td>';        /* Paramters for each picture */
  if coordinates eq "No GPS" or
    upcase("&includeGPSlinks.") eq "NO" then
    put '<img src=' picture ' width="' width
      '" height="' height '"></td></tr><tr>';
  else put '<a href="http://maps.google.com/maps?f='
    'q&source=s_q&hl=en&geocode=&q=' coordinates
    '&aq=&ie=UTF8&t=f&z=16&vpsrc=0&ecpose=' coordinates
    ',1025.22,0,0,0"> <img src=' picture ' width="'
    width '" height="' height '"></a></td></tr><tr>';
  if upcase("&afterpicturenumber.") eq "YES" then
   put '<td width="' width '" style="color:blue">' title
   '<br/><br/></td><td width="' width '"align="right"> '
   _n_ '<br/><br/></td></tr>';
  else put '       <td width="' width '" style="color:blue">'
   title '<br/><br/></td></tr>';
  if done then                              /* Final wrapup */
    put /  '</table>'
        / '</body>'
        / '</html>';
run;
```