Paper 024-2013

# The Ins and Outs of Web-Based Data with SAS®

Bill McNeill, SAS Institute Inc., Cary, NC

## ABSTRACT

Do you have data on the web that you want to integrate with SAS®? This paper explains how you can obtain web data, process it, and export it back to the web. Examples use existing features, such as the SOAP and XSL procedures, the SAS® XML Mapper application, and the XMLV2 LIBNAME engine, along with two new features: the XMLV2 LIBNAME engine AUTOMAP= option and the JSON procedure. The AUTOMAP= option allows for creation of default XMLMap files within SAS. The JSON procedure exports SAS data sets in JSON format to an external file. And if you need to write free-form JSON output, forget the SAS PUT statements, the JSON procedure supports free-form JSON output as well.

## INTRODUCTION

This paper contains working examples that use various sources of data on the Internet to create and display a custom itinerary for my SAS® Global Forum experience. I will assemble data from the Internet to display the SAS Global Forum schedule, directions from my hotel to the conference, and the weather for each day that I am at the conference. From there, more example code  demonstrates how the tools can potentially be used to surface the SAS data on the Internet in HTML, XML, and JSON formats. All the SAS programs mentioned in this paper and associated files created by the SAS programs are contained in a zip file available from the SAS Support site at: http://support.sas.com/saspresents.

## HTML INPUT (METHOD 1)

HyperText Markup Language (HTML) is the old man for displaying information in a web browser. It has been around since almost the beginning of the Internet. Its markup tags contained within angle brackets are most likely familiar to you:

```
<h2>Schedule at a Glance</h2>
```

This line produces the text "Schedule at a Glance" in a format specified for the H2 tag, which is the #2 heading. Somewhere else in the HTML document is a definition of how to format a #2 heading. This is all great for output in a web browser, but what if the HTML contains data you wish to use in a SAS session? How is this data accessed? One method is via the SAS FILENAME statement using the URL file access method. Therefore, accessing the web page is as simple as:

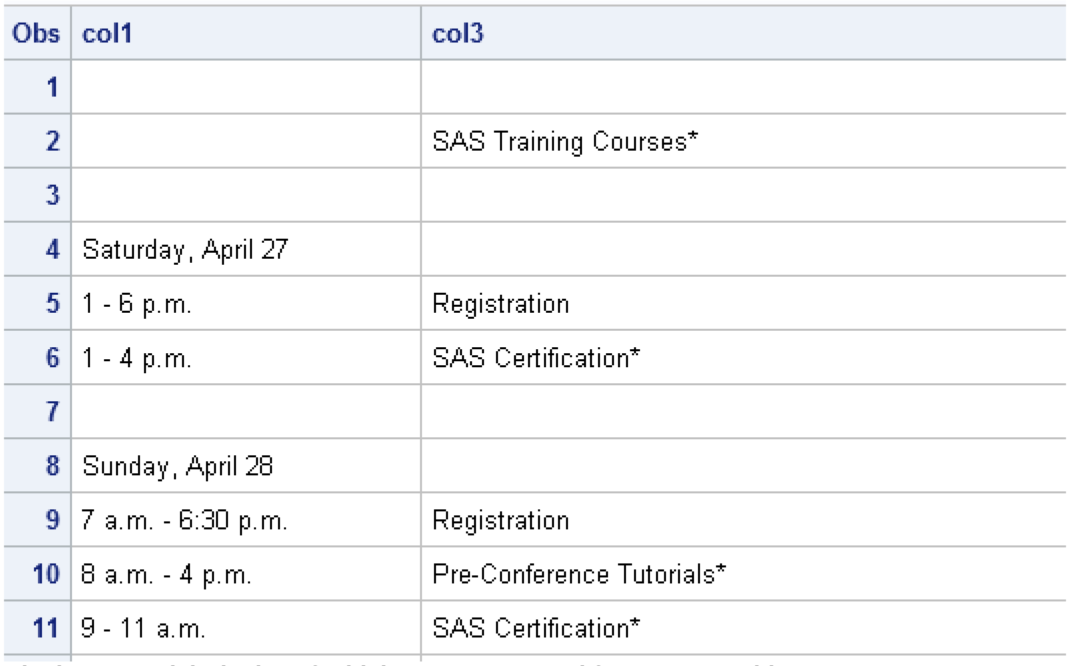
```
filename foo url
   'http://support.sas.com/events/sasglobalforum/2013/glance.html';
```

This statement assigns the fileref "foo" to the specified HTML file. The URL file access method allows for reading a file via a network connection to a running server. Unfortunately, there are many lines of HTML prior to the lines of data of interest. For example, here is a snippet of the start of the HTML file:

```
!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"><!-- InstanceBegin
template="/Templates/sgf2013-subpage.dwt" codeOutsideHTMLIsLocked="false" -->
<head>
<!-- InstanceBeginEditable name="Meta" -->
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<meta name="Description" content="High level preliminary agenda for SAS Global
Forum 2013 conference to be held April 28 to May 1, 2013 in San Francisco,
California, USA. " />
```

```
<meta name="Keywords" content="SAS Global Forum 2013, SUGI, Schedule, Schedule-
at-a-glance" />
<meta name="date" content="2012-24-09" />
<meta name="Robots" content="index,follow"/>
<!-- InstanceEndEditable -->
<!-- InstanceBeginEditable name="doctitle" -->
<title>SAS Global Forum 2013 Schedule at a Glance</title>
<!-- InstanceEndEditable -->
<link href="/events/sasglobalforum/2013/css/sgf.css" rel="stylesheet"
type="text/css" />
```

The data of interest actually starts on line 443 in this file. Note that this line number is specific to this file. Another HTML file with table data will most likely be very different. Therefore, custom DATA step code might be needed to obtain the data from each web page. A paper by colleague Rick Langston details a general SAS program for extracting HTML table data from any web page (Langston 2009). This was a very good start, but modifications were still needed to properly obtain the desired data from this specific web page. The modified program is named readHtmlTables.sas. This program contains over 300 lines of SAS DATA step and macro code. The result of running the code is a data set containing the table data.

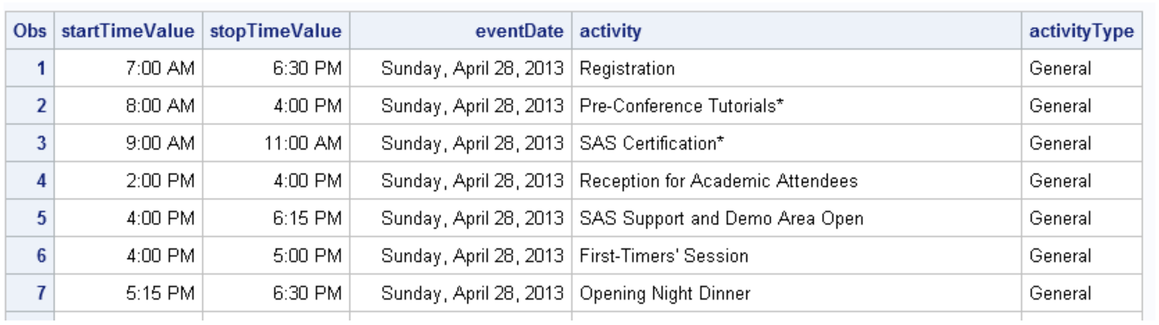


| Obs | col1 | col3 |
|---|---|---|
| 1 | | |
| 2 | | SAS Training Courses* |
| 3 | | |
| 4 | Saturday, April 27 | |
| 5 | 1 - 6 p.m. | Registration |
| 6 | 1 - 4 p.m. | SAS Certification* |
| 7 | | |
| 8 | Sunday, April 28 | |
| 9 | 7 a.m. - 6:30 p.m. | Registration |
| 10 | 8 a.m. - 4 p.m. | Pre-Conference Tutorials* |
| 11 | 9 - 11 a.m. | SAS Certification* |

**Display 1. Partial Display of Initial Data Set Created from HTML Table Data**

The data is not yet in the desired format. Running another custom SAS program, processHtmlTable.sas, gets the data in the following desired format:

- The blank observations have been eliminated.
- Only the main conference days, Sunday through Wednesday, activities are included.
- Each observation now includes the date as a SAS date value.
- The activity times are now SAS time values.
- An activity type was added to the observation.



| Obs | startTimeValue | stopTimeValue | eventDate | activity | activityType |
|---|---|---|---|---|---|
| 1 | 7:00 AM | 6:30 PM | Sunday, April 28, 2013 | Registration | General |
| 2 | 8:00 AM | 4:00 PM | Sunday, April 28, 2013 | Pre-Conference Tutorials* | General |
| 3 | 9:00 AM | 11:00 AM | Sunday, April 28, 2013 | SAS Certification* | General |
| 4 | 2:00 PM | 4:00 PM | Sunday, April 28, 2013 | Reception for Academic Attendees | General |
| 5 | 4:00 PM | 6:15 PM | Sunday, April 28, 2013 | SAS Support and Demo Area Open | General |
| 6 | 4:00 PM | 5:00 PM | Sunday, April 28, 2013 | First-Timers' Session | General |
| 7 | 5:15 PM | 6:30 PM | Sunday, April 28, 2013 | Opening Night Dinner | General |

**Display 2: Partial Display of Data Set Based on the HTML Table Data in the Desired Format**

## HTML INPUT (METHOD 2)

Before proceeding to the next step, I would like to demonstrate another method of obtaining the data from the HTML table. This method involves the XSL procedure, which is included with Base SAS® for SAS 9.2 and later. The XSL procedure allows for transforming an XML file. XSL stands for Extensible Stylesheet Language. The XSL procedure passes the input XML file and a user-created XSL stylesheet to the XSL transformation program. The program applies the instructions in the XSL stylesheet to the input XML file to create an output XML file. But wait, the input file is HTML. Correct, but HTML is similar enough in format to XML and the XSL language powerful enough that transforming HTML is also possible.

The first attempt to transform the HTML code produces a number of errors as the website makes use of JavaScript. Because the data of interest in the HTML is contained between the <TABLE> elements, a small SAS DATA step program was written to extract just the HTML table data to a file. This DATA step code along with the running of the XSL procedure is included in the xslHtmlTable.sas file.

With just the data contained in the HTML table, the XSL transformation works without complaint. The whole XSL stylesheet is called htmlTable.xsl. I am greatly indebted to my colleague Carlotta Hicks for creating this stylesheet for me. While teaching XSL is outside the scope of this paper, a brief summary of the transformation is in the file xslDescription.txt. Here is the code to run the XSL stylesheet with the XSL procedure:

```
filename in  'data\justTable.html';
filename xsl 'sasuser\htmlTable.xsl';
filename out 'data\schedule.xml';

proc xsl in=in xsl=xsl out=out;
run;
```

Here is the Saturday data in the HTML file being transformed:

```
      <table border="0" cellspacing="0" cellpadding="0">
        <tr>
          <td valign="top"><p><strong>Saturday,    April 27</strong></p></td>
          <td valign="top"></td>
          <td valign="top"></td>
        </tr>
        <tr>
          <td valign="top"><p>1    - 6 p.m.</p></td>
          <td valign="top"></td>
          <td valign="top"><p>Registration </p></td>
        </tr>
        <tr>
          <td valign="top"><p>1    - 4 p.m.</p></td>
          <td valign="top"></td>
          <td valign="top"><p><a target="_blank" href="testing.html">SAS
Certification</a>*</p></td>
        </tr>
        <tr>
          <td valign="top"></td>
          <td valign="top"></td>
          <td valign="top"></td>
        </tr>
        <tr>
      </table>
```

This is the Saturday data in the XML file created by the XSL transformation. The full XML file is in the file named schedule.xml.

```
   <Schedule>
      <Date>Saturday,    April 27</Date>
      <Activities>
         <Time>1    - 6 p.m.</Time>
         <Event>Registration </Event>
         <Time>1    - 4 p.m.</Time>
         <Event>SAS    Certification*</Event>
      </Activities>
   </Schedule>
```

Use the XMLV2 LIBNAME engine to read this file into a SAS data set.

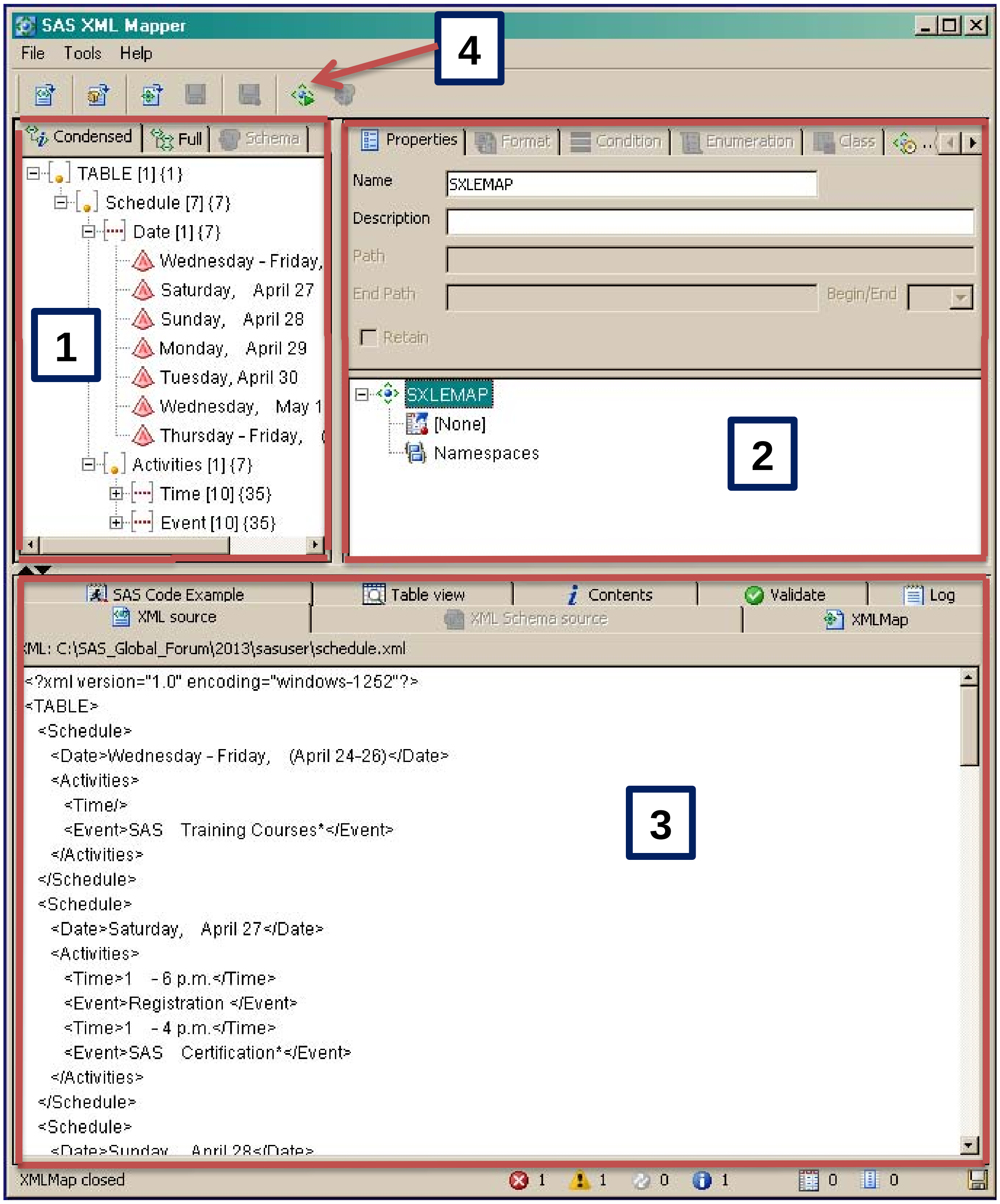
```
libname sched xmlv2 xmlfileref=out;
```

Unfortunately, the LIBNAME statement results in the following error messages:

```
ERROR: XML data is not in a format supported natively by the XML libname
       engine. Files of this type may require an XMLMap to be input properly.
ERROR: Error in the LIBNAME statement.
```

This error is due to the XML file multilevel hierarchical structure that cannot be translated in a straight forward fashion to the rectangular format that SAS data sets require. To solve this problem, an XMLMap file is needed. This can be created with the SAS XML Mapper application (introduced in SAS 8.2). This is a stand-alone application available in the Base SAS install or as a free download from the SAS Support website.

With the SAS XML Mapper application running, selecting **File->Open XML…** allows for opening the XML file. The Information Pane (section 3 in Display 3) displays the XML file. The XML Primary Pane (section 1 in Display 3) shows the hierarchy of the data in the file.



```
TABLE [1] {1}
    Schedule [7] {7}
        Date [1] {7}
            Wednesday - Friday,
            Saturday,    April 27
            Sunday,    April 28
            Monday,    April 29
            Tuesday, April 30
            Wednesday,    May 1
            Thursday - Friday,
        Activities [1] {7}
            Time [10] {35}
            Event [10] {35}
```

Name: SXLEMAP
Description:
Path:
End Path: Begin/End:
Retain

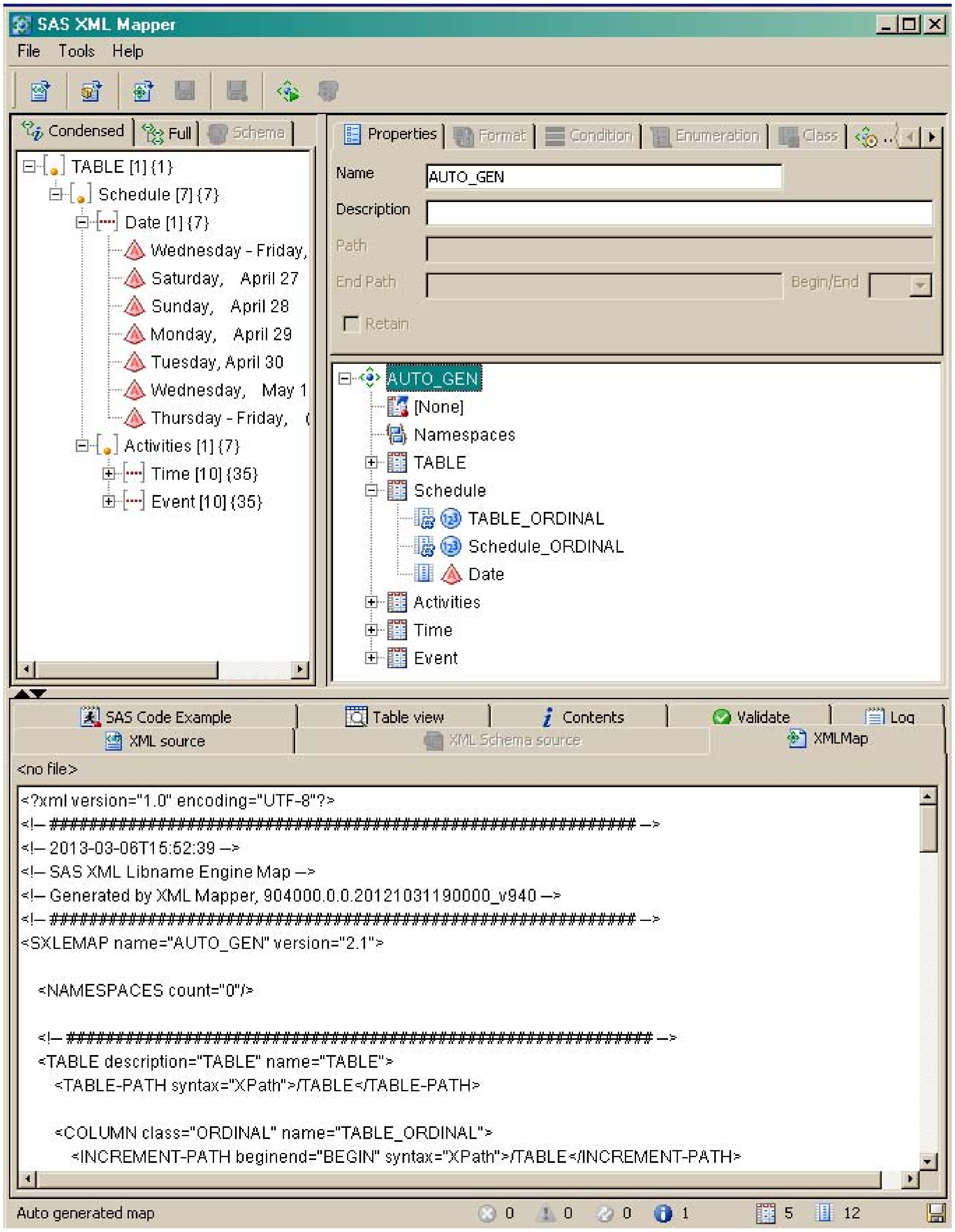
```
SXLEMAP
    [None]
    Namespaces
```

```
XML: C:\SAS_Global_Forum\2013\sasuser\schedule.xml
<?xml version="1.0" encoding="windows-1252"?>
<TABLE>
  <Schedule>
    <Date>Wednesday - Friday,    (April 24-26)</Date>
    <Activities>
      <Time/>
      <Event>SAS    Training Courses*</Event>
    </Activities>
  </Schedule>
  <Schedule>
    <Date>Saturday,    April 27</Date>
    <Activities>
      <Time>1    - 6 p.m.</Time>
      <Event>Registration </Event>
      <Time>1    - 4 p.m.</Time>
      <Event>SAS    Certification*</Event>
    </Activities>
  </Schedule>
  <Schedule>
    <Date>Sunday,    April 28</Date>
```

1 1 0 1 0 0

**Display 3: Display of XML File in SAS XML Mapper Application**

To create the SAS data sets, use the icon (pointed to by the arrow labeled 4 in Display 3) or select **Tools->Automap using XML** to automatically generate an XMLMap file from the input XML file. The resulting XMLMap file can be viewed in the Information Pane when the XMLMap tab is clicked.

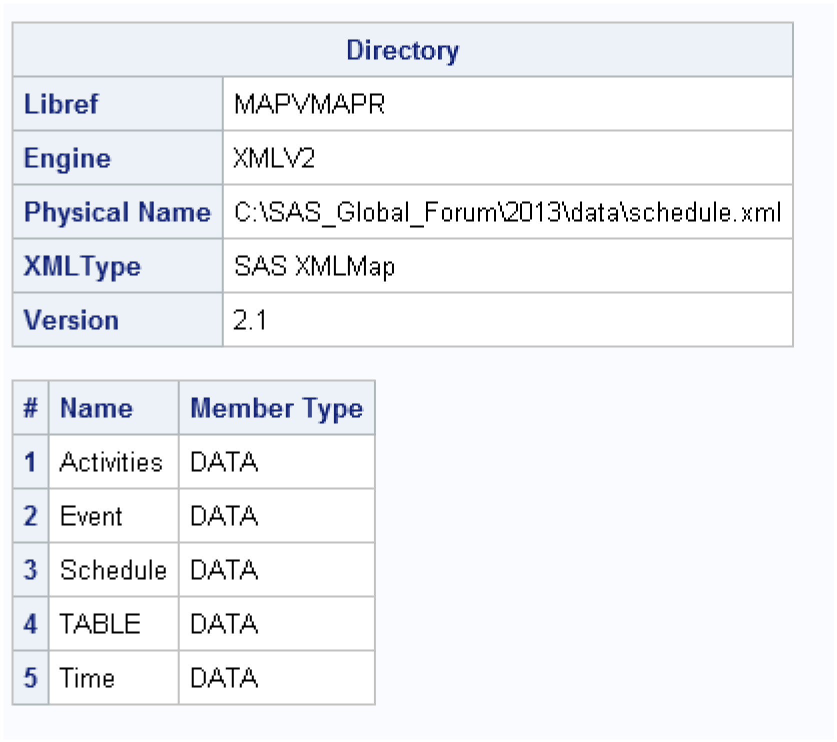


```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ############################################################ -->
<!-- 2013-03-06T15:52:39 -->
<!-- SAS XML Libname Engine Map -->
<!-- Generated by XML Mapper, 904000.0.0.20121031190000_v940 -->
<!-- ############################################################ -->
<SXLEMAP name="AUTO_GEN" version="2.1">

  <NAMESPACES count="0"/>

  <!-- ############################################################ -->
  <TABLE description="TABLE" name="TABLE">
    <TABLE-PATH syntax="XPath">/TABLE</TABLE-PATH>

    <COLUMN class="ORDINAL" name="TABLE_ORDINAL">
      <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/TABLE</INCREMENT-PATH>
```

**Display 4: Display of XMLMap File Created in SAS XML Mapper Application**

The XML Primary Pane shows the data sets that will be generated by using the XMLMap file with the XMLV2 LIBNAME engine. The **Date** entry is expanded to show the data that will be contained in the SAS data set. In the Information Pane is the XMLMap file. Save this file by selecting **File->Save XMLMap as…** Now it can be used by the XMLV2 LIBNAME engine to create the data sets.

```
libname mapVmapr xmlv2 xmlfileref=out xmlmap="data\schedule.map";
```



| Directory | |
|---|---|
| Libref | MAPVMAPR |
| Engine | XMLV2 |
| Physical Name | C:\SAS_Global_Forum\2013\data\schedule.xml |
| XMLType | SAS XMLMap |
| Version | 2.1 |

| # | Name | Member Type |
|---|---|---|
| 1 | Activities | DATA |
| 2 | Event | DATA |
| 3 | Schedule | DATA |
| 4 | TABLE | DATA |
| 5 | Time | DATA |

**Display 5: List of the Data Sets Created from the XML File Using the XMLMap File**

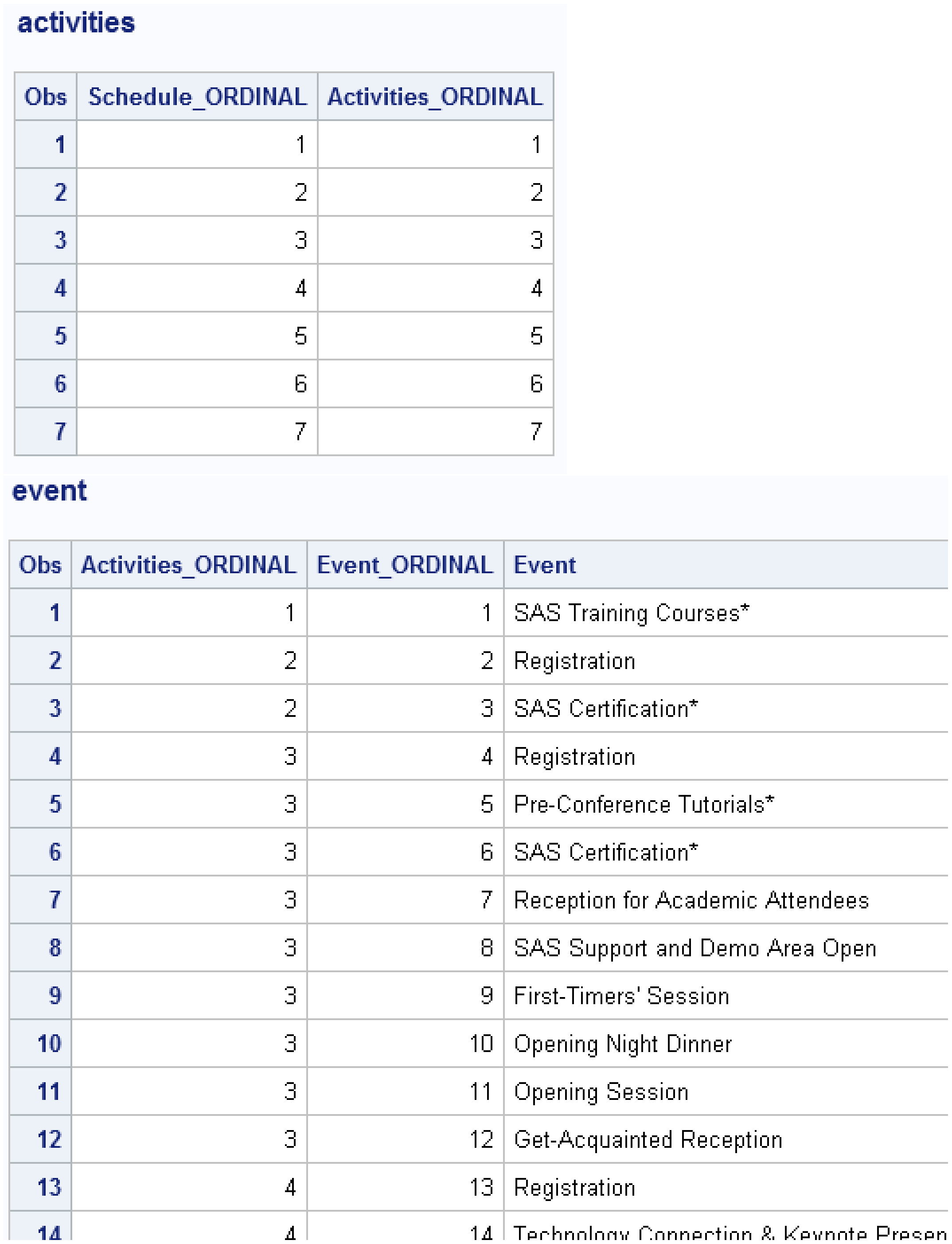
As you can see from the DATASETS procedure output (Display 5), a number of data sets are created from the XML file.

A slightly different and easier way of creating an XMLMap file is available in the second maintenance release for SAS 9.3. It is the AUTOMAP= option. If all you want to do is automatically generate the XMLMap file, this option allows you to create the XMLMap file in Base SAS. There is no need to run the SAS XML Mapper application. The AUTOMAP= option accepts two values: REPLACE and REUSE. The REPLACE value overwrites an XMLMap file of the same name at the same location. The REUSE value checks, and if a file of the specified name is available, that file  is used. If not, a new XMLMap file is created. Therefore, in this case with the directions.xml file, instead of running the SAS XML Mapper application, all that is needed is the following code:

```
filename tempMap temp;
libname sched xmlv2 xmlfileref=out xmlmap=tempMap automap=replace;
```

The FILENAME statement creates a temporary location for the XMLMap file. AUTOMAP=REPLACE results in the generation of a new XMLMap file and its usage by the LIBNAME statement.

To understand what the SAS XML Mapper application does, it helps to look at the data sets. (The images of the longer data sets show only a portion of the data set.) Note that the TABLE data set is not shown because it only includes one observation of the single variable table_ordinal.



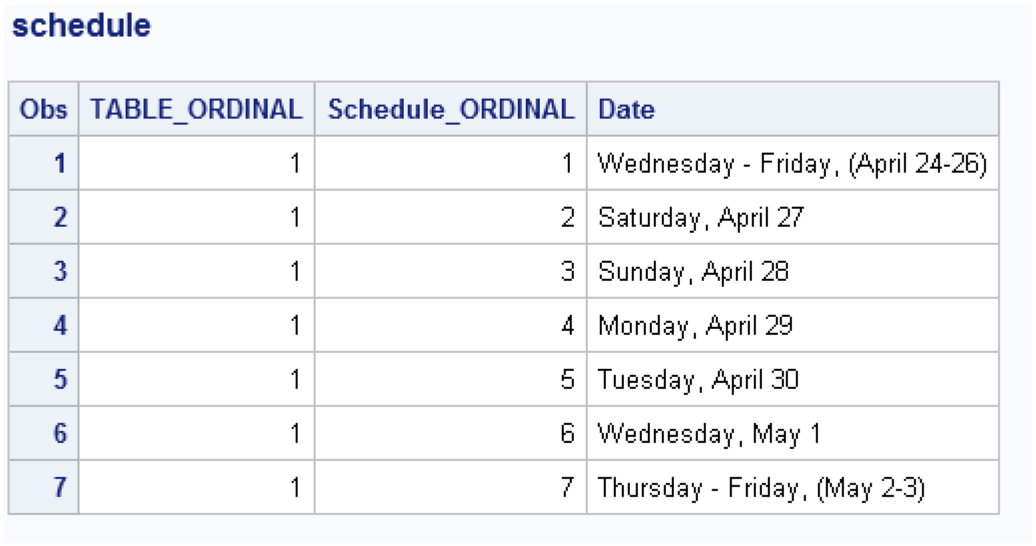
**activities**

| Obs | Schedule_ORDINAL | Activities_ORDINAL |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 2 | 2 |
| 3 | 3 | 3 |
| 4 | 4 | 4 |
| 5 | 5 | 5 |
| 6 | 6 | 6 |
| 7 | 7 | 7 |

**event**

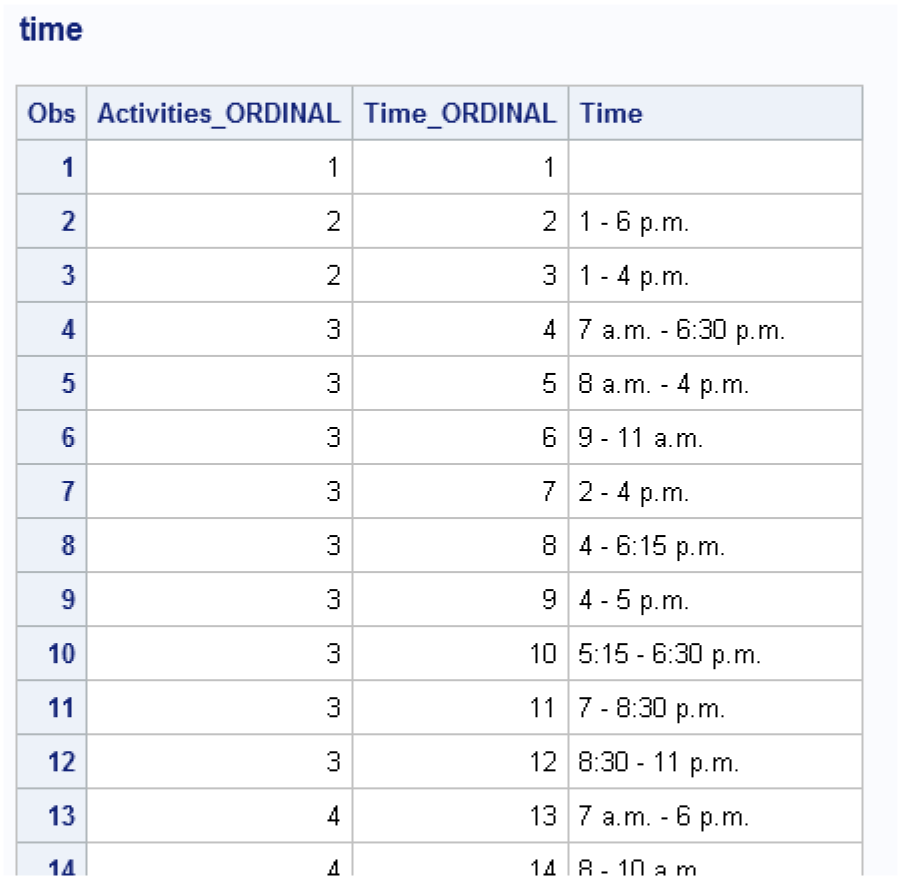
| Obs | Activities_ORDINAL | Event_ORDINAL | Event |
|---|---|---|---|
| 1 | 1 | 1 | SAS Training Courses* |
| 2 | 2 | 2 | Registration |
| 3 | 2 | 3 | SAS Certification* |
| 4 | 3 | 4 | Registration |
| 5 | 3 | 5 | Pre-Conference Tutorials* |
| 6 | 3 | 6 | SAS Certification* |
| 7 | 3 | 7 | Reception for Academic Attendees |
| 8 | 3 | 8 | SAS Support and Demo Area Open |
| 9 | 3 | 9 | First-Timers' Session |
| 10 | 3 | 10 | Opening Night Dinner |
| 11 | 3 | 11 | Opening Session |
| 12 | 3 | 12 | Get-Acquainted Reception |
| 13 | 4 | 13 | Registration |
| 14 | 4 | 14 | Technology Connection & Keynote Presen |

**Display 6: Output from the ACTIVITIES and EVENT Data Sets**



**schedule**

| Obs | TABLE_ORDINAL | Schedule_ORDINAL | Date |
|---|---|---|---|
| 1 | 1 | 1 | Wednesday - Friday, (April 24-26) |
| 2 | 1 | 2 | Saturday, April 27 |
| 3 | 1 | 3 | Sunday, April 28 |
| 4 | 1 | 4 | Monday, April 29 |
| 5 | 1 | 5 | Tuesday, April 30 |
| 6 | 1 | 6 | Wednesday, May 1 |
| 7 | 1 | 7 | Thursday - Friday, (May 2-3) |



**time**

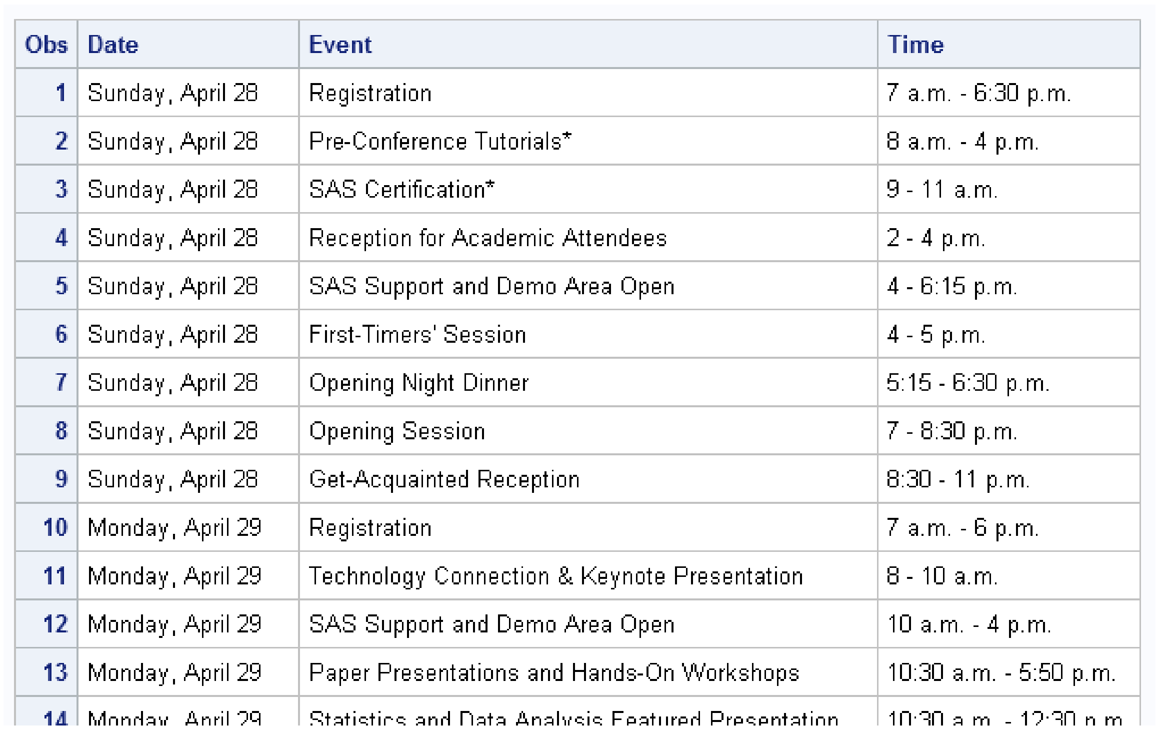
| Obs | Activities_ORDINAL | Time_ORDINAL | Time |
|---|---|---|---|
| 1 | 1 | 1 | |
| 2 | 2 | 2 | 1 - 6 p.m. |
| 3 | 2 | 3 | 1 - 4 p.m. |
| 4 | 3 | 4 | 7 a.m. - 6:30 p.m. |
| 5 | 3 | 5 | 8 a.m. - 4 p.m. |
| 6 | 3 | 6 | 9 - 11 a.m. |
| 7 | 3 | 7 | 2 - 4 p.m. |
| 8 | 3 | 8 | 4 - 6:15 p.m. |
| 9 | 3 | 9 | 4 - 5 p.m. |
| 10 | 3 | 10 | 5:15 - 6:30 p.m. |
| 11 | 3 | 11 | 7 - 8:30 p.m. |
| 12 | 3 | 12 | 8:30 - 11 p.m. |
| 13 | 4 | 13 | 7 a.m. - 6 p.m. |
| 14 | 4 | 14 | 8 - 10 a.m |

**Display 7: Output from the SCHEDULE and TIME Data Sets**

The SAS XML Mapper application adds the *_ORDINAL variables as a way to piece together the relationship of the data in the tables. This allows using the SQL procedure to combine the data into a single data set:

```
proc sql;
create table work.xslSchedule as (
    select dates.date,
           events.event,
           times.time
    from sched.schedule   as dates,
         sched.activities as acts,
         sched.event      as events,
         sched.time       as times
    where ( ( (dates.schedule_ordinal GE 3) and
              (dates.schedule_ordinal LE 6)     )              and
            dates.schedule_ordinal   = acts.activities_ordinal   and
            acts.activities_ordinal = events.activities_ordinal and
            acts.activities_ordinal = times.activities_ordinal  and
            times.time_ordinal      = events.event_ordinal)
    );
quit;
```

Note that the WHERE clause allows for schedule_ordinal values between 3 and 6 only. This reduces the data to Sunday – Wednesday of the conference, which are the days that I will be in attendance. The data set is shown in the following partial listing of the data.



| Obs | Date | Event | Time |
|---|---|---|---|
| 1 | Sunday, April 28 | Registration | 7 a.m. - 6:30 p.m. |
| 2 | Sunday, April 28 | Pre-Conference Tutorials* | 8 a.m. - 4 p.m. |
| 3 | Sunday, April 28 | SAS Certification* | 9 - 11 a.m. |
| 4 | Sunday, April 28 | Reception for Academic Attendees | 2 - 4 p.m. |
| 5 | Sunday, April 28 | SAS Support and Demo Area Open | 4 - 6:15 p.m. |
| 6 | Sunday, April 28 | First-Timers' Session | 4 - 5 p.m. |
| 7 | Sunday, April 28 | Opening Night Dinner | 5:15 - 6:30 p.m. |
| 8 | Sunday, April 28 | Opening Session | 7 - 8:30 p.m. |
| 9 | Sunday, April 28 | Get-Acquainted Reception | 8:30 - 11 p.m. |
| 10 | Monday, April 29 | Registration | 7 a.m. - 6 p.m. |
| 11 | Monday, April 29 | Technology Connection & Keynote Presentation | 8 - 10 a.m. |
| 12 | Monday, April 29 | SAS Support and Demo Area Open | 10 a.m. - 4 p.m. |
| 13 | Monday, April 29 | Paper Presentations and Hands-On Workshops | 10:30 a.m. - 5:50 p.m. |
| 14 | Monday, April 29 | Statistics and Data Analysis Featured Presentation | 10:30 a.m. - 12:30 p.m. |

**Display 8: Partial Output Listing from Combining the Data Sets from the XML File**

The XSL method took less custom SAS code to write and produced a data set similar to the DATA step code earlier, but the drawback is learning XSL.

## WEB SERVICES

Web services refer to sites on the Internet where you can request specific information. The requested information is returned usually in either XML or JSON format. Access to web services on the Internet can be accomplished with the HTTP and SOAP procedures in Base SAS.

### HTTP

To enhance the itinerary data, I will get directions to the conference from the hotel where I am staying. I am close enough so that I can walk to the conference each day. The Google Maps API allows me to get walking directions by supplying the start and ending addresses (Google Maps API 2013). I just looked these up manually from the Internet. The Google Maps API Web Services contains various HyperText Transfer Protocol (HTTP) interfaces. The one of interest is the Directions API. By using the HTTP procedure (introduced in SAS 9.2) and the instructions at the Google Directions API, https://developers.google.com/maps/documentation/directions/, I can create a SAS program, getHttpXml.sas, to get the walking directions from the hotel to the conference.

```
%let startAddress=50 Third Street San Francisco, CA;
%let endAddress=800 Howard St, San Francisco, CA;
%let tranportMode=walking;

%let dirStart=?origin=&startAddress;
%let dirEnd=%nrstr(&destination=)&endAddress;
%let dirURL=http://maps.googleapis.com/maps/api/directions/;
%let dirDataType=xml;
%let dirMode=%nrstr(&mode=)&tranportMode;
%let dirSensor=%nrstr(&sensor=false);
%let dirFullURL=&dirURL.&dirDataType.&dirStart.&dirEnd.&dirMode.&dirSensor;
filename directs 'data\directions.xml';
proc http out=directs
          url="&dirFullURL"
         method="GET"
         ct="application/x-www-form-urlencoded";
      run;
```

The results are returned in the file directions.xml. I can now use the XMLV2 LIBNAME engine to convert the XML file to SAS data sets. Note that I take advantage of the AUTOMAP= option in the LIBNAME statement.

```
filename tempMap Temp;
libname walkDirs xmlv2 xmlfileref=directs xmlmap=tempMap automap=replace;
```

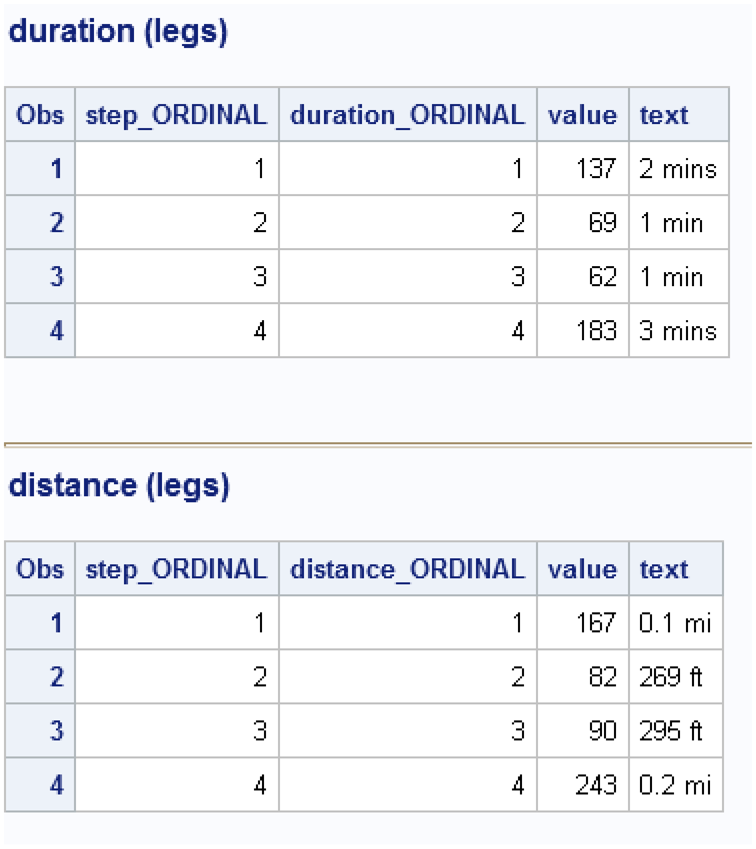
With the data sets created, it is time to assemble them into two data sets that contain only the information I need for the itinerary. The first data set is a summary of the directions. The straightforward SQL procedure code is available in getHttpXml.sas. The resulting data set is:

**directions summary**

| Obs | totalTime | totalDistance | copyrights | walkWarning | hotelAddress | sgfAddress |
|---|---|---|---|---|---|---|
| 1 | 8 mins | 0.4 mi | Map data ©2013 Google, Sanborn | Walking directions are in beta. Use caution – This route may be missing sidewalks or pedestrian paths. | 50 3rd Street, San Francisco, CA 94103, USA | 800 Howard Street, San Francisco, CA 94103, USA |

**Display 9: List of the DIRECTIONS SUMMARY Data Set**

Creating the data set with the specific directions  is a bit more involved. Here are the data sets before being joined into one.



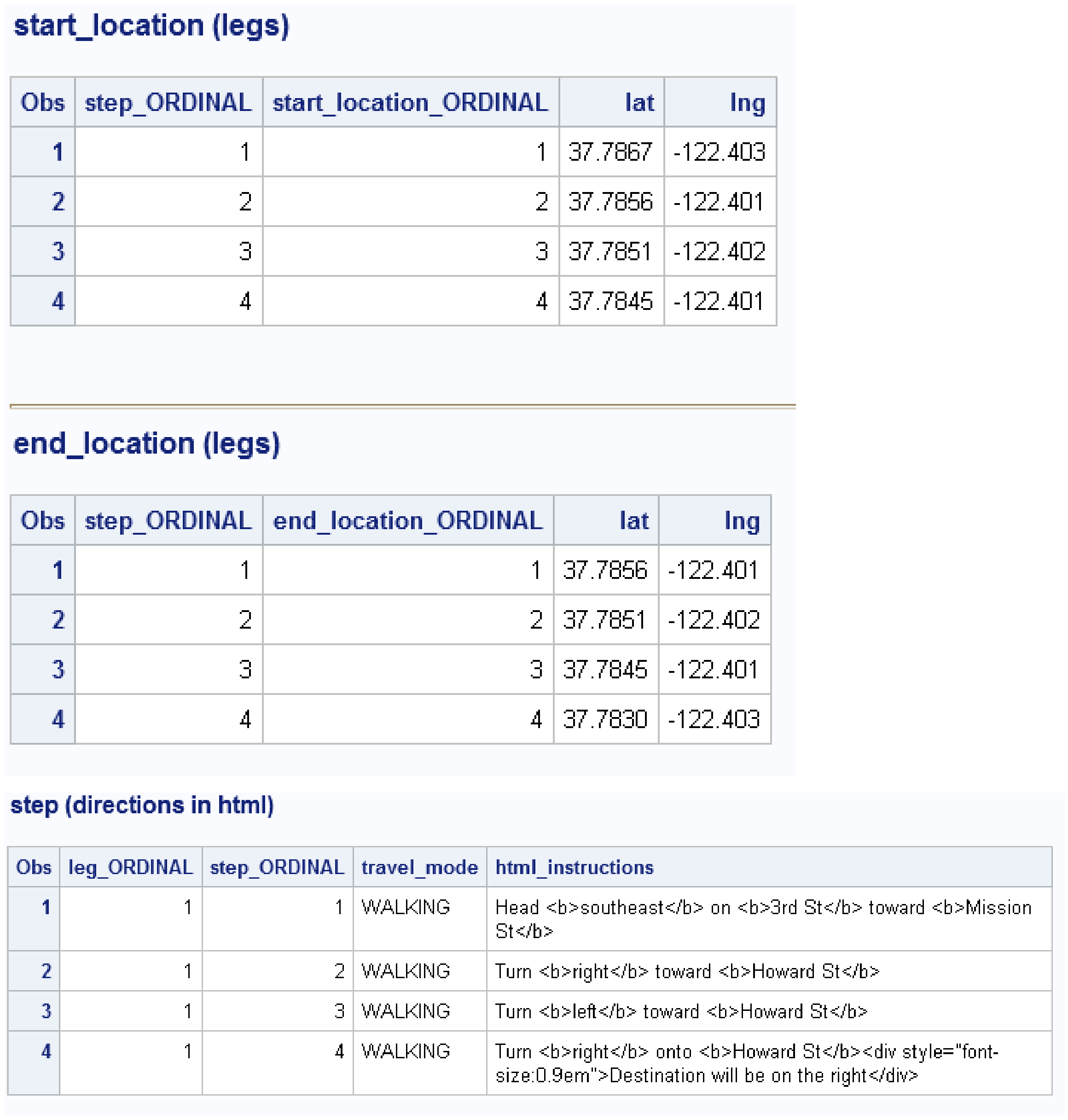
duration (legs)

| Obs | step_ORDINAL | duration_ORDINAL | value | text |
|---|---|---|---|---|
| 1 | 1 | 1 | 137 | 2 mins |
| 2 | 2 | 2 | 69 | 1 min |
| 3 | 3 | 3 | 62 | 1 min |
| 4 | 4 | 4 | 183 | 3 mins |

distance (legs)

| Obs | step_ORDINAL | distance_ORDINAL | value | text |
|---|---|---|---|---|
| 1 | 1 | 1 | 167 | 0.1 mi |
| 2 | 2 | 2 | 82 | 269 ft |
| 3 | 3 | 3 | 90 | 295 ft |
| 4 | 4 | 4 | 243 | 0.2 mi |

**Display 10: Listings of the DURATION and DISTANCE Data Sets**



start_location (legs)

| Obs | step_ORDINAL | start_location_ORDINAL | lat | lng |
|---|---|---|---|---|
| 1 | 1 | 1 | 37.7867 | -122.403 |
| 2 | 2 | 2 | 37.7856 | -122.401 |
| 3 | 3 | 3 | 37.7851 | -122.402 |
| 4 | 4 | 4 | 37.7845 | -122.401 |

end_location (legs)

| Obs | step_ORDINAL | end_location_ORDINAL | lat | lng |
|---|---|---|---|---|
| 1 | 1 | 1 | 37.7856 | -122.401 |
| 2 | 2 | 2 | 37.7851 | -122.402 |
| 3 | 3 | 3 | 37.7845 | -122.401 |
| 4 | 4 | 4 | 37.7830 | -122.403 |

step (directions in html)

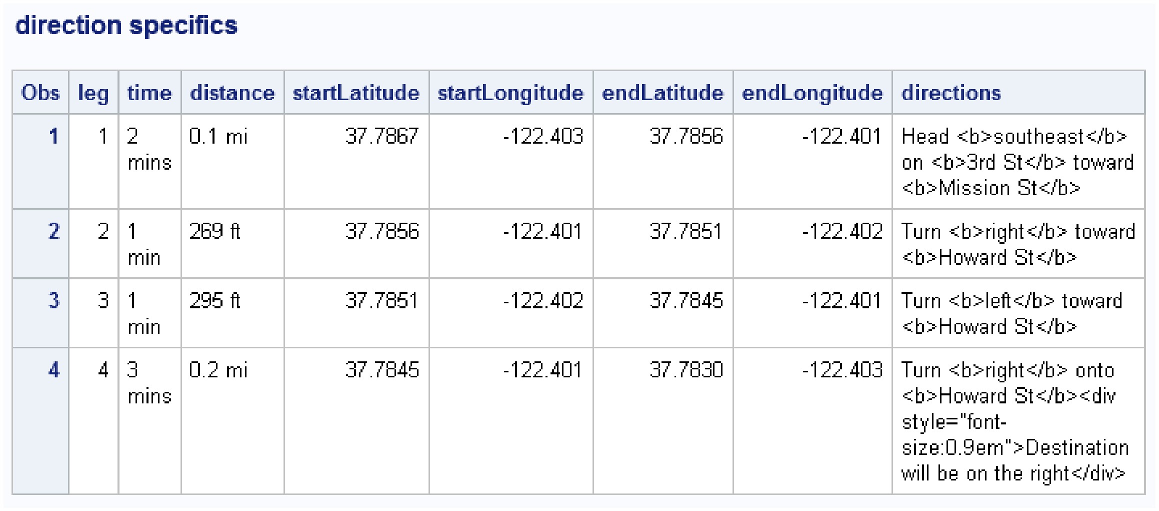
| Obs | leg_ORDINAL | step_ORDINAL | travel_mode | html_instructions |
|---|---|---|---|---|
| 1 | 1 | 1 | WALKING | Head <b>southeast</b> on <b>3rd St</b> toward <b>Mission St</b> |
| 2 | 1 | 2 | WALKING | Turn <b>right</b> toward <b>Howard St</b> |
| 3 | 1 | 3 | WALKING | Turn <b>left</b> toward <b>Howard St</b> |
| 4 | 1 | 4 | WALKING | Turn <b>right</b> onto <b>Howard St</b><div style="font-size:0.9em">Destination will be on the right</div> |

**Display 11: Listings of the START_LOCATION, END_LOCATION, and STEP Data Sets**

Due to the hierarchy of the XML file, the data sets contain ordinal values tying the data sets together. In this case, all the tables have the step_ordinal variable in common. Joining on this value results in the needed organization of the data (as seen in getHttpXml.sas).

```
proc sql;
create table work.dirSpec as (
   select step.step_ordinal       as leg,
          dur.text                as time,
          dis.text                as distance,
          start.lat               as startLatitude,
          start.lng               as startLongitude,
          end.lat                 as endLatitude,
          end.lng                 as endLongitude,
          step.html_instructions  as directions
   from (walkDirs.duration       as dur,
         walkDirs.distance       as dis,
         walkDirs.start_location as start,
         walkDirs.end_location   as end,
         walkDirs.step           as step)
   where (dur.step_ordinal = dis.step_ordinal   and
          dur.step_ordinal = start.step_ordinal and
          dur.step_ordinal = end.step_ordinal   and
          dur.step_ordinal = step.step_ordinal      )
   );
   quit;
```

The resulting data set is shown below.

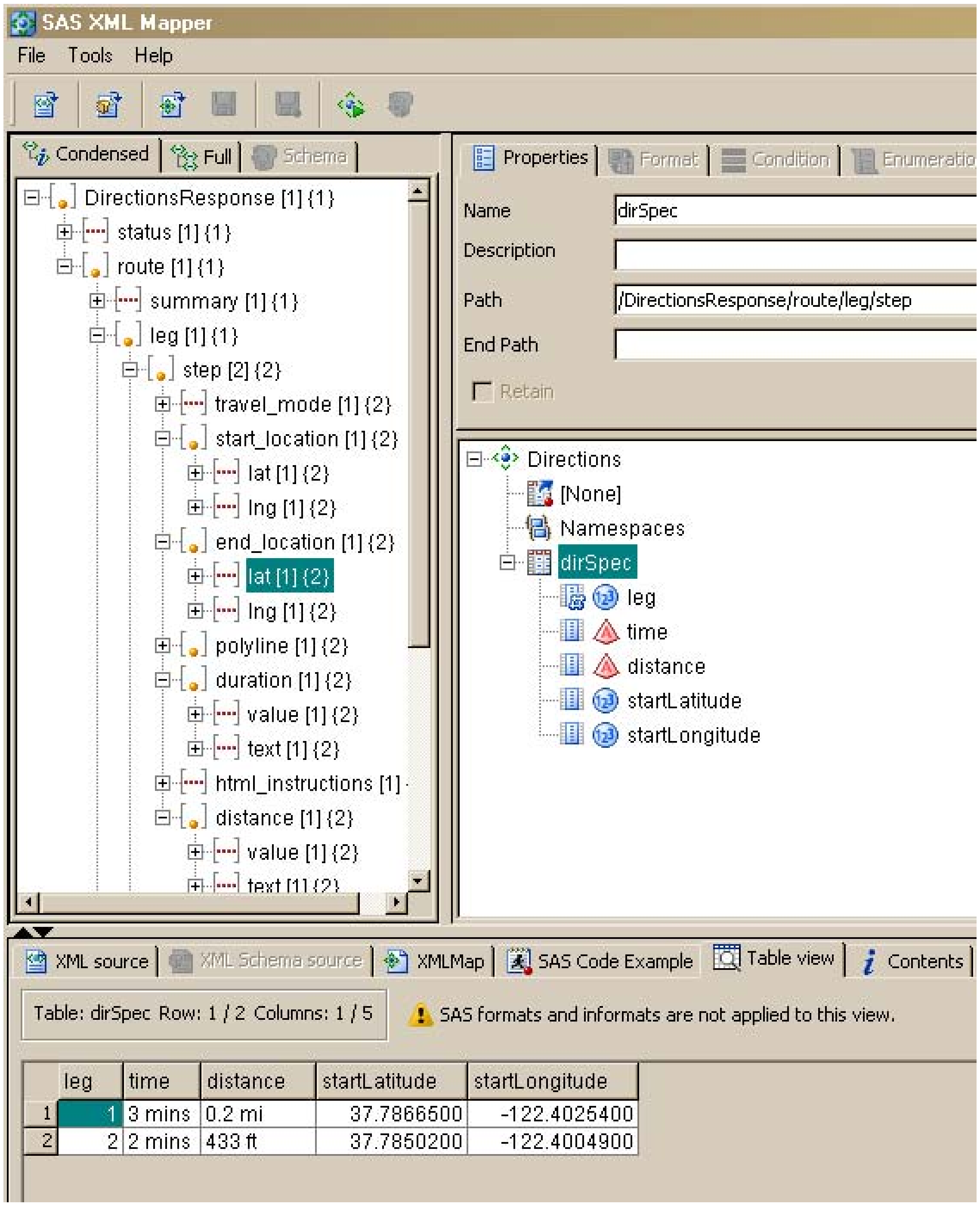


direction specifics

| Obs | leg | time | distance | startLatitude | startLongitude | endLatitude | endLongitude | directions |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 mins | 0.1 mi | 37.7867 | -122.403 | 37.7856 | -122.401 | Head <b>southeast</b> on <b>3rd St</b> toward <b>Mission St</b> |
| 2 | 2 | 1 min | 269 ft | 37.7856 | -122.401 | 37.7851 | -122.402 | Turn <b>right</b> toward <b>Howard St</b> |
| 3 | 3 | 1 min | 295 ft | 37.7851 | -122.402 | 37.7845 | -122.401 | Turn <b>left</b> toward <b>Howard St</b> |
| 4 | 4 | 3 mins | 0.2 mi | 37.7845 | -122.401 | 37.7830 | -122.403 | Turn <b>right</b> onto <b>Howard St</b><div style="font-size:0.9em">Destination will be on the right</div> |

**Display 12: Listing of the DIRSPEC Data Set**

Another way to create this table is by taking further advantage of the SAS XML Mapper application. Creating a custom XMLMap file for the desired output eliminates the SQL procedure code shown above, which creates the DIRSPEC data set. As shown previously, the directions.xml file is loaded into the SAS XML Mapper application. But, instead of using the automatic mapping feature, only the needed data is selected. Start by expanding the tree in the XML Primary Pane, **route->leg->step**. From having just gone through this process with the SQL procedure, you now know that it is the entries under this path that make up the desired data set. In the XMLMap Primary Pane, the **SXLEMAP** entry is already selected. Go up to the **Name** field and change the name to “Directions” (without the quotation marks). Now click the **step** entry in the XML Primary Pane and drag it on top of the **Directions** entry in the bottom half of the XMLMap Primary Pane. This results in **step** appearing under the **Directions** entry. In the bottom of the XMLMap Primary Pane frame, click the **step** entry, go to the top of that pane, and, in the **Name** field, change the name to “dirSpec” (without the quotation marks). All the names need to be changed to match the names in the data set created with the SQL procedure. Next, to get the **leg** values matched up properly (which was done by the SQL WHERE clause), create the ordinal value for **leg**. To do this, use the right mouse button to click **dirSpec** and, in the pop-up menu, select **Create row count ordinal**. This creates the **_dirSpec_ID** entry.



Name: dirSpec
Description:
Path: /DirectionsResponse/route/leg/step
End Path:

Table: dirSpec Row: 1 / 2 Columns: 1 / 5

SAS formats and informats are not applied to this view.

| | leg | time | distance | startLatitude | startLongitude |
|---|---|---|---|---|---|
| 1 | 1 | 3 mins | 0.2 mi | 37.7866500 | -122.4025400 |
| 2 | 2 | 2 mins | 433 ft | 37.7850200 | -122.4004900 |

**Display 13: View of the SAS XML Mapper Application Creating the XMLMap File**

Note the display of the partially complete output format at the bottom of the display. Rename it to “leg”.

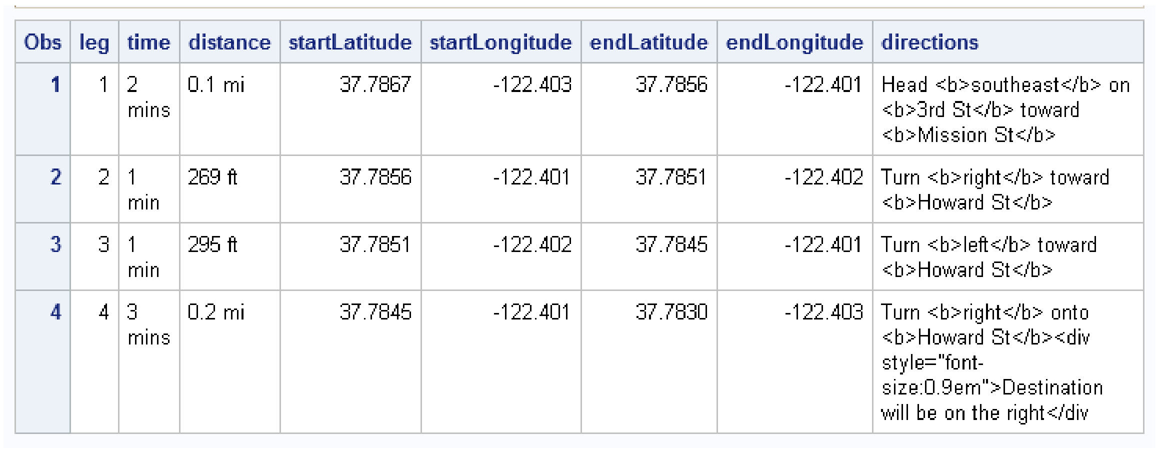
The following steps are the same but with different entries in the tree view of the XML file: expand the specified path, click the specified entry, drag and drop the entry onto the **dirSpec** entry in the XMLMap Primary Pane, and rename the entry. Before doing that, at the top of the Information Pane, click the tab labeled **Table view**. This shows the output as the XMLMap is created. Here are the steps, based on what was done above, for the rest of the needed data:

1. Expand **duration**, drag **text**, and rename as “time”.

2. Expand **distance**, drag **text**, and rename as “distance”.
3. Expand **start_location**, drag **lat**, and rename as “startLatitude”.
4. From **start_location**, also drag **lng**, and rename as “startLongitude”.
5. Expand **end_location**, drag **lat**, and rename as “endLatitude”.
6. From **end_location**, also drag **lng**, and rename as “endLongitude”.
7. Click **html_instructions**, drag **html_instructions**, and rename as “directions”.

As you can see in the Information Pane, the data from the XML file looks just like the data set created by the SQL procedure. Now, to verify the same results as the SQL procedure code, save the XMLMap file by selecting **File->Save XMLMap as…** and use via the XMLV2 LIBNAME engine with the XMLMAP= option.

```
libname walkDir2 xmlv2 xmlfileref=directs xmlmap="data\directions.map";
proc print data=walkDir2.dirSpec;
run;
```



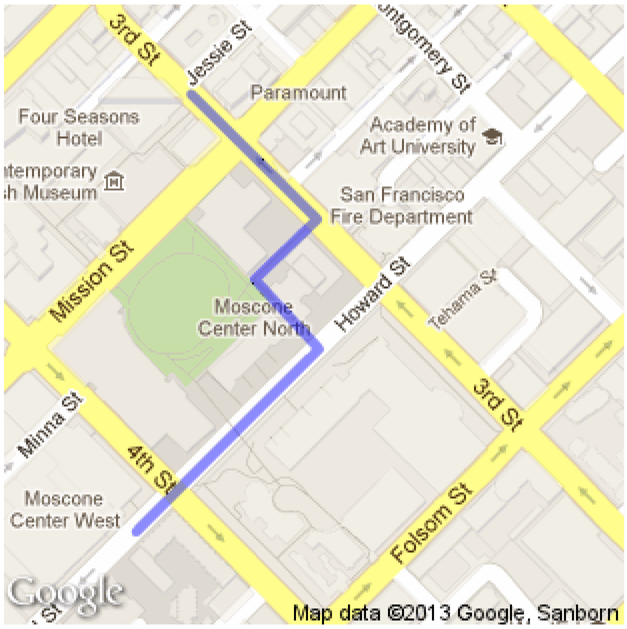
| Obs | leg | time | distance | startLatitude | startLongitude | endLatitude | endLongitude | directions |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 mins | 0.1 mi | 37.7867 | -122.403 | 37.7856 | -122.401 | Head <b>southeast</b> on <b>3rd St</b> toward <b>Mission St</b> |
| 2 | 2 | 1 min | 269 ft | 37.7856 | -122.401 | 37.7851 | -122.402 | Turn <b>right</b> toward <b>Howard St</b> |
| 3 | 3 | 1 min | 295 ft | 37.7851 | -122.402 | 37.7845 | -122.401 | Turn <b>left</b> toward <b>Howard St</b> |
| 4 | 4 | 3 mins | 0.2 mi | 37.7845 | -122.401 | 37.7830 | -122.403 | Turn <b>right</b> onto <b>Howard St</b><div style="font-size:0.9em">Destination will be on the right</div |

**Display 14: Listing of the DIRSPEC Data Set**

My original intention was to be done with this step at this point. But when I read through the Google API documentation, I found that to use the Google directions, I was required by the terms of the API to include a Google map image of the route. After assembling the needed URL, this is the code to create the map image of the route:

```
filename mapImage "output\map.png";
proc http out=mapImage
          url="%superq(fullMapURL)"
          method="GET"
          ct="application/x-www-form-urlencoded";
run;
```

The latitude and longitude values for mapping the routes are taken from the data set. The HTTP procedure is again used to obtain the data. In this case, the data is an image file showing the route from the hotel to the conference. The full code is available in getHttpXml.sas.



**Display 15: Image of Walking Route from the Hotel to the Conference**

## SOAP

Because I will be walking, it would be good to know the weather forecast while I am at the conference. This data is easily obtained from the National Weather Service via a Simple Object Access Protocol (SOAP) web service (National Oceanic and Atmospheric Administration (NOAA) National Weather Service (NWS) National Digital Web Service 2013). A SOAP web service works on a messaging system. You send a message to the web service in a format specified by the web service. The message contains a request for data and any needed parameters for specifying the data. The web service responds with a message containing the desired data. The messages are in XML. The description of the request and response messages is available in a Web Services Description Language (WSDL) file. This is another XML file that details the requirements of the messages. While the WSDL file can be read and deciphered, I find it easier to use a third-party tool like SoapUI (SoapUI 2013). When this tool is pointed to the API website, it allows you to select from a list the message for the desired data. Once selected, it produces the SOAP request message with placeholders for any message parameters.

In this case, the NDFDgenByDay function is used to obtain the weather forecast for the four days I will be at SAS Global Forum. The function requires six parameters, the latitude and longitude of the location, the start date and the number of days of forecasts, the type of units to return, and the format of the data. The hotel's latitude and longitude coordinates were obtained from the walking directions in the previous section of this paper. This allows for programmatically adding the coordinates to the SOAP request message. Note that because this paper is being written in March and the National Weather Service Web Service does not offer forecasts that far in advance (eight weeks), the examples of the forecasts are for San Francisco during the first week of March.

The message format for requesting data from the NDFDgenByDay function basically looks like:

```
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ndf="http://graphical.weather.gov/xml/DWMLgen/wsdl/ndfdXML.wsdl">
   <soapenv:Header/>
   <soapenv:Body>
      <ndf:NDFDgenByDay
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <latitude xsi:type="xsd:decimal">?</latitude>
         <longitude xsi:type="xsd:decimal">?</longitude>
         <startDate xsi:type="xsd:date">?</startDate>
         <numDays xsi:type="xsd:integer">?</numDays>
         <Unit xsi:type="dwml:unitType"
xmlns:dwml="http://graphical.weather.gov/xml/DWMLgen/schema/DWML.xsd">?</Unit>
         <format xsi:type="dwml:formatType"
xmlns:dwml="http://graphical.weather.gov/xml/DWMLgen/schema/DWML.xsd">?</format
>
      </ndf:NDFDgenByDay>
   </soapenv:Body>
</soapenv:Envelope>
```

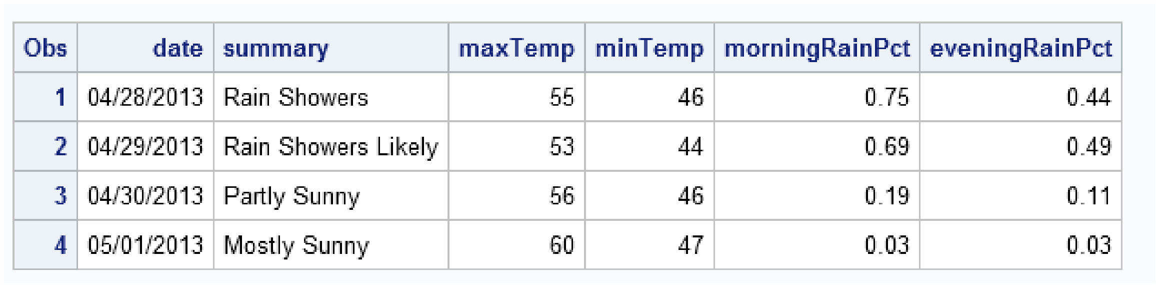
Note the six question marks for the parameter placeholders. Those are the values I need to supply to get the desired data. The code is contained in the getSoapXml.sas file

Once the SOAP message is created, the SOAP procedure (introduced in SAS 9.2) is used to send the request message and obtain the response message. Here is the code to accomplish this exchange:

```
proc soap in=request
          out=response
          url="http://graphical.weather.gov/xml/SOAP_server/ndfdXMLserver.php"

soapaction="http://graphical.weather.gov/xml/DWMLgen/wsdl/ndfdXML.wsdl#NDFDgenB
yDay"
          proxyhost="xxxxxx.xxxxxx.sas.com"
          proxyport=80
          ;
run;
```

Note that in my case, a proxy host name and port number is needed to communicate with the web service. Your site might not require these options. For more information, see the SOAP procedure in the *Base SAS Procedures Guide*.

The response message, soapOutput.xml, also in XML, supplies the information requested in the request message. The response file contains hierarchical data and therefore needs an XMLMap file to be read into SAS as a data set. A default XMLMap file is created from the response XML file and is used to create the data sets. After some DATA step manipulation, the data sets are ready to be fit into the desired format with the SQL procedure. (See getSoapXml.sas.) Here is the final data set with all the weather data.



| Obs | date | summary | maxTemp | minTemp | morningRainPct | eveningRainPct |
|---|---|---|---|---|---|---|
| 1 | 04/28/2013 | Rain Showers | 55 | 46 | 0.75 | 0.44 |
| 2 | 04/29/2013 | Rain Showers Likely | 53 | 44 | 0.69 | 0.49 |
| 3 | 04/30/2013 | Partly Sunny | 56 | 46 | 0.19 | 0.11 |
| 4 | 05/01/2013 | Mostly Sunny | 60 | 47 | 0.03 | 0.03 |

**Display 16: Listing of the Daily Weather Statistics**

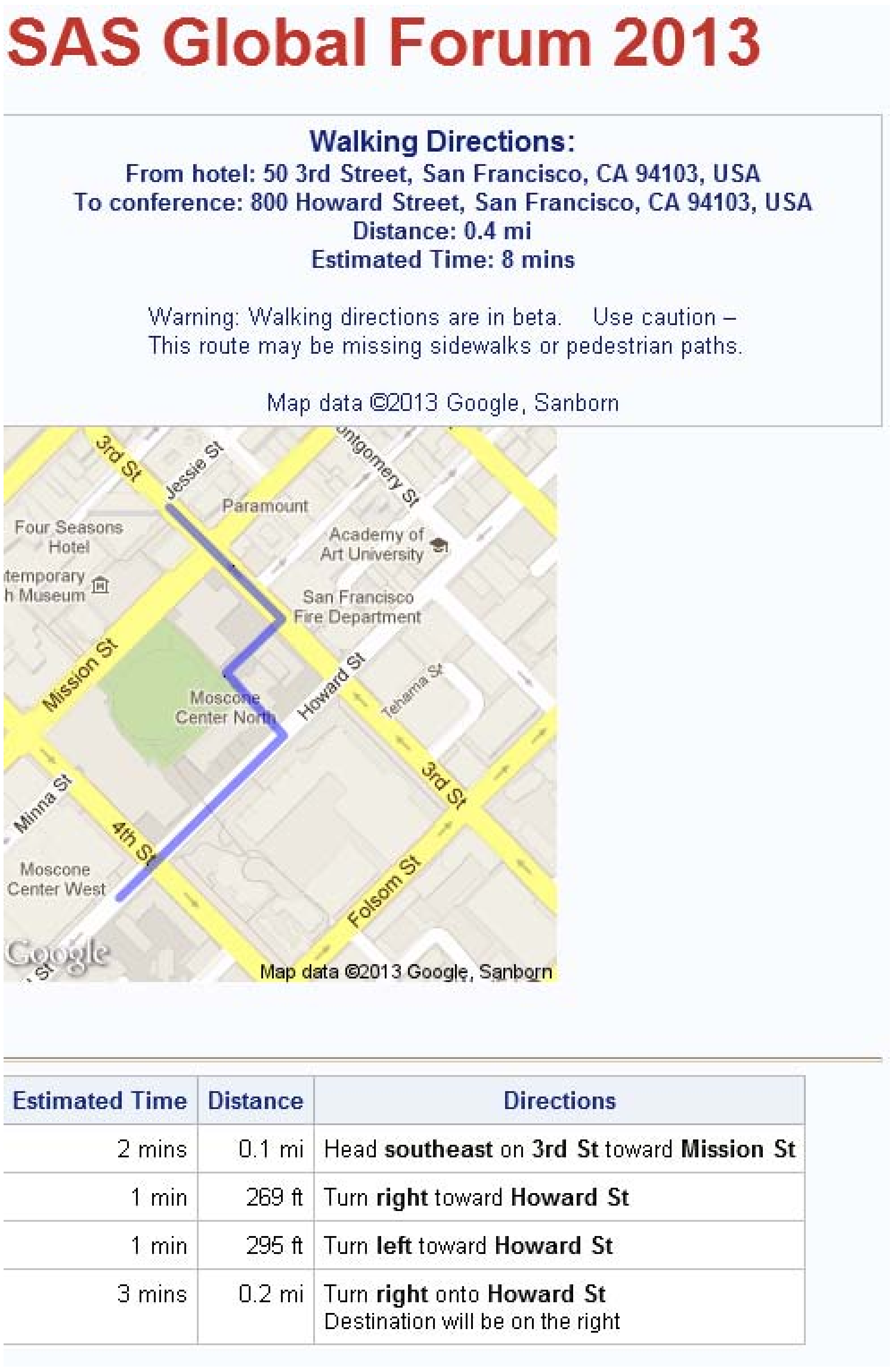
The final piece of the desired output is enhancing the schedule to include required, desired, and recommended activities. Required activities are things like presenting this paper, being at the Base SAS Demo Station during my assigned times, and presenting a Super Demo. The recommended activities are things that, as a SAS employee, I should attend if at all possible like the Opening Session and the Technology Connection. Finally, things that I desire to do were added. This is mainly presentations of other papers that interest me. To make sure that the activities stand out on the daily schedule, I have color coded them: red for Required, yellow for Recommended, and green for Desired.

## HTML OUTPUT

All the desired data is collected into SAS data sets. Now it is time to make the data available on the Internet. First up is to create the itinerary report. This will be a static HTML page created with the REPORT procedure. This code is available in the displaySchedule.sas program. The HTML output file, full2013Schedule.html, is created by simply wrapping the REPORT procedure code with the ODS HTML statements.

```
ods html path="output/" body="full2013Schedule.html";
proc report …

run;

ods html close;
```

What follows are the screen shots of the first two sections of the finished report (with made up weather data for the actual conference dates).

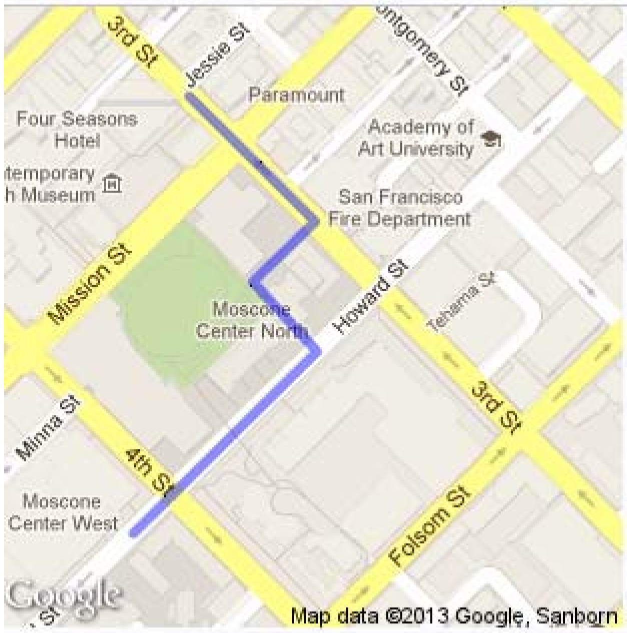


SAS Global Forum 2013

**Walking Directions:**
From hotel: 50 3rd Street, San Francisco, CA 94103, USA
To conference: 800 Howard Street, San Francisco, CA 94103, USA
Distance: 0.4 mi
Estimated Time: 8 mins

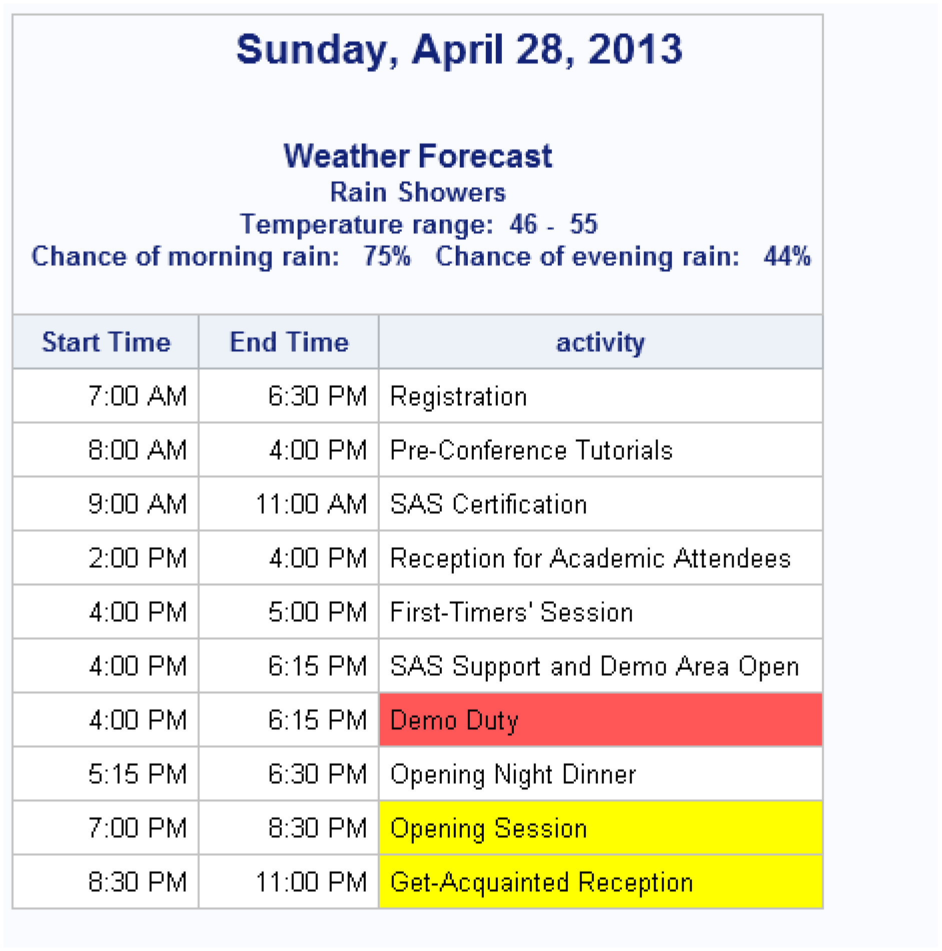
Warning: Walking directions are in beta. Use caution –
This route may be missing sidewalks or pedestrian paths.

Map data ©2013 Google, Sanborn



| Estimated Time | Distance | Directions |
|---|---|---|
| 2 mins | 0.1 mi | Head **southeast** on **3rd St** toward **Mission St** |
| 1 min | 269 ft | Turn **right** toward **Howard St** |
| 1 min | 295 ft | Turn **left** toward **Howard St** |
| 3 mins | 0.2 mi | Turn **right** onto **Howard St**<br>Destination will be on the right |

**Display 17: First Part of the Itinerary Web Page**



Sunday, April 28, 2013

Weather Forecast
Rain Showers
Temperature range: 46 - 55
Chance of morning rain: 75% Chance of evening rain: 44%

| Start Time | End Time | activity |
|---|---|---|
| 7:00 AM | 6:30 PM | Registration |
| 8:00 AM | 4:00 PM | Pre-Conference Tutorials |
| 9:00 AM | 11:00 AM | SAS Certification |
| 2:00 PM | 4:00 PM | Reception for Academic Attendees |
| 4:00 PM | 5:00 PM | First-Timers' Session |
| 4:00 PM | 6:15 PM | SAS Support and Demo Area Open |
| 4:00 PM | 6:15 PM | Demo Duty |
| 5:15 PM | 6:30 PM | Opening Night Dinner |
| 7:00 PM | 8:30 PM | Opening Session |
| 8:30 PM | 11:00 PM | Get-Acquainted Reception |

**Display 18: Display of the Sunday Itinerary**

## XML OUTPUT

As seen on the input side, there is more to web data than HTML. While surfacing the itinerary in an HTML page is good for viewing as noted earlier, getting data from HTML can be difficult. The data collected for the itinerary is easier to consume from the web when in XML format. Exporting any or all of the data sets with XML is fairly simple. Here is an example from exportData.sas of sending the daily weather to the XML file weatherOut.xml:

```
libname wthrOut xmlv2 "output\weatherOut.xml";
data wthrOut.confWeather;
set work.weather;
run;
```

This results in the following XML code:

```
<?xml version="1.0" encoding="windows-1252" ?>
<TABLE>
   <CONFWEATHER>
      <date>2013-04-28</date>
      <summary>Rain Showers</summary>
      <maxTemp>55</maxTemp>
      <minTemp>46</minTemp>
      <morningRainPct>0.75</morningRainPct>
      <eveningRainPct>0.44</eveningRainPct>
   </CONFWEATHER>
   <CONFWEATHER>
      <date>2013-04-29</date>
      <summary>Rain Showers Likely</summary>
      <maxTemp>53</maxTemp>
      <minTemp>44</minTemp>
      <morningRainPct>0.69</morningRainPct>
      <eveningRainPct>0.49</eveningRainPct>
   </CONFWEATHER>
   <CONFWEATHER>
      <date>2013-04-30</date>
      <summary>Partly Sunny</summary>
      <maxTemp>56</maxTemp>
      <minTemp>46</minTemp>
      <morningRainPct>0.19</morningRainPct>
      <eveningRainPct>0.11</eveningRainPct>
   </CONFWEATHER>
   <CONFWEATHER>
      <date>2013-05-01</date>
      <summary>Mostly Sunny</summary>
      <maxTemp>60</maxTemp>
      <minTemp>47</minTemp>
      <morningRainPct>0.03</morningRainPct>
      <eveningRainPct>0.03</eveningRainPct>
   </CONFWEATHER>
</TABLE>
```

For the XMLV2 engine, the outer most element defaults to the named TABLE. The next level element is the SAS member name specified in the DATA statement. The elements under that are the variable names from the SAS data set. In this rectangular format, the XML can be read back into a SAS data set without the need for an XMLMap file.

## JSON OUTPUT

Another popular method for presenting data on the Internet is in JSON format. New in SAS 9.4, the JSON procedure allows for exporting SAS data sets in JSON format. To output the same weather data set to an output file in JSON format would be done with the following code in exportData.sas:

```
proc json out="output\weatherOut.json" pretty nosastags;
export work.weather;
run;
```

The syntax in this example is straight forward. The only thing that really needs explaining are the two options: PRETTY and NOSASTAGS. By default, the JSON output is formatted to a single line, essentially for streaming the data. For this example, the PRETTY option makes the output data easier to read. The NOSASTAGS option suppresses metadata that is supplied by SAS that appears before the exported data. The metadata is suppressed to mimic the format style seen in some web services that surface JSON formatted data.

```
[
  {
    "date": "04/28/2013",
    "summary": "Rain Showers",
    "maxTemp": 55,
    "minTemp": 46,
    "morningRainPct": 0.75,
    "eveningRainPct": 0.44
  },
  {
    "date": "04/29/2013",
    "summary": "Rain Showers Likely",
    "maxTemp": 53,
    "minTemp": 44,
    "morningRainPct": 0.69,
    "eveningRainPct": 0.49
  },
  {
    "date": "04/30/2013",
    "summary": "Partly Sunny",
    "maxTemp": 56,
    "minTemp": 46,
    "morningRainPct": 0.19,
    "eveningRainPct": 0.11
  },
  {
    "date": "05/01/2013",
    "summary": "Mostly Sunny",
    "maxTemp": 60,
    "minTemp": 47,
    "morningRainPct": 0.03,
    "eveningRainPct": 0.03
  }
]
```

The JSON procedure also includes the ability to export multiple data sets to the same JSON file, use data set options, and write free-form JSON data. This next example shows all of these by adding some free-form JSON to the output:

```
proc json out="output\weatherOut2.json" pretty nosastags;
write value "Daily Weather";
  write open array;
  write value "Sunday";
    export personal.weather(where=(date="28APR2013"d));
  write value "Monday";
    export personal.weather(where=(date="29APR2013"d));
  write value "Tuesday";
    export personal.weather(where=(date="30APR2013"d));
  write value "Wednesday";
    export personal.weather(where=(date="01MAY2013"d));
  write close;
run;
```

The output of this code from exportData.sas is:

```
{
  "Daily Weather": [
    "Sunday",
    {
      "date": "04/28/2013",
      "summary": "Rain Showers",
      "maxTemp": 55,
      "minTemp": 46,
      "morningRainPct": 0.75,
      "eveningRainPct": 0.44
    },
    "Monday",
    {
      "date": "04/29/2013",
      "summary": "Rain Showers Likely",
      "maxTemp": 53,
      "minTemp": 44,
      "morningRainPct": 0.69,
      "eveningRainPct": 0.49
    },
    "Tuesday",
    {
      "date": "04/30/2013",
      "summary": "Partly Sunny",
      "maxTemp": 56,
      "minTemp": 46,
      "morningRainPct": 0.19,
      "eveningRainPct": 0.11
    },
    "Wednesday",
    {
      "date": "05/01/2013",
      "summary": "Mostly Sunny",
      "maxTemp": 60,
      "minTemp": 47,
      "morningRainPct": 0.03,
      "eveningRainPct": 0.03
    }
  ]
}
```

Writing JSON or XML to a file is one thing, but what about setting up a web service like the one used earlier in this paper to obtain data from the Google Map Directions and the National Weather Service and pass the data over the Internet? SAS can do that with SAS® BI Web Services. Going into the details of SAS BI Web Services is outside the scope of this paper. But briefly, by using stored processes, which are SAS programs registered with the web service, SAS data can be surfaced via SOAP or HTTP interfaces. The surfaced data can be in XML format, or starting in SAS 9.3, JSON format. See the documentation at the SAS Support website for more information about SAS BI Web Services, http://support.sas.com/documentation/cdl/en/wbsvcdg/62759/PDF/default/wbsvcdg.pdf.

## CONCLUSION

SAS has a number of tools to access data on the Internet as well as to make SAS data accessible to the Internet. I have supplied examples that, with some very minor tweaking, could be running on your SAS session within minutes. This gives you a starting point to begin exploring these tools. I mentioned SAS BI Web Services for surfacing SAS data to the Internet. Finally, I showed what is new in SAS 9.3 (the AUTOMAP= option) and what is just around the corner in SAS 9.4 (the JSON procedure). The next time you see some interesting data on the Internet, try using the tools and techniques shown here to get that data into and out of SAS.

## REFERENCES

Google Maps API. (2013). Google Developers website. Available at https://developers.google.com/maps/. Accessed February 22, 2013.

Langston, Rick. 2009. “Creating SAS® Data Sets from HTML Table Definitions.” *Proceedings of the SAS® Global Forum 2009 Conference*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings09/052-2009.pdf.

National Oceanic and Atmospheric Administration (NOAA) National Weather Service (NWS) National Digital Web Service. (2013). Forecast Database (NDFD) Simple Object Access Protocol (SOAP) Web Service website. Available at http://graphical.weather.gov/xml/. Accessed February 7, 2013.

SoapUI. (2013). Obtained SoapUI software. Available at http://www.soapui.org/. Accessed February 12, 2013.

## ACKNOWLEDGMENTS

The author would like to thank the following for their input on the paper: Beth Ayres, Lisa Brown, Mike Carney, Thomas Cox, Carlotta Hicks, MySa Ho, Karen Hoffman, Rick Langston, Zach Marshall, Chevell Parker, Deanna Warner, and Sandy McNeill.

## RECOMMENDED READING

- *Base SAS® Procedures Guide*
- *SAS® XML LIBNAME Engine: User's Guide*
- *SAS® Macro Language: Reference*
- *SAS® Language Reference: Concepts*
- *SAS® Language Reference: Statements*

## CONTACT INFORMATION

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

Bill McNeill
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Bill.McNeill@sas.com