Ottawa Area SAS Users Society

www.OASUS.ca

Books-by-Users
Web Development with SAS® by Example (Third Edition) Frederick E. Pratter

"Come out of the desert of ignorance to the OASUS of knowledge"

Ottawa Area SAS Users Society

www.OASUS.ca

Three books in one!

- Introduction and history of the Internet and HTML
- Overview of SAS Enterprise Business Intelligence
- How to develop SAS applications for the Web

Ottawa Area SAS Users Society

www.OASUS.ca

Introduction and history of the Internet and HTML

- Chapter 1: SAS and the Internet
  - Introduction to Internet technologies
  - History of the Internet
  - How to set up a web server
- Chapter 2: Introduction to HTML
  - Introduction to how HTML works
  - HTML versus XHTML
  - Style Sheets and Forms

Background, very well written and informative

Ottawa Area SAS Users Society

www.OASUS.ca

Overview of SAS Enterprise Business Intelligence (1)

- Chapter 5: Remote Access to SAS Data
  - Client/Server Computing
  - SAS/Share and remote access
  - Introduction to IOM
- Chapter 6: Access to Relational Databases Using SAS
  - SAS/Access to Relational DBMS (Oracle)
  - SQL Pass-Through
  - ODBC, OLE DB, Access for PC Files

Ottawa Area SAS Users Society

www.OASUS.ca

Overview of SAS Enterprise Business Intelligence (2)

- Chapter 9: SAS Enterprise Business Intelligence

Introduction to SAS EBI, covered in detail later on

- SAS IOM servers (Metadata, Workspace, Stored Process, OLAP)
- Application programs: Mid-Tier (Information Delivery Portal, Web Report Studio, Stored Process Web Application, BI Dashboard)
- Application programs: Client (Management Console, OLAP Cube Studio, Information Map Studio, Enterprise Guide
- Client/Server Architecture (SAS/Connect, SAS/Share, Integration Technologies)
- Java Web Applications
- Installation of SAS EBI

Ottawa Area SAS Users Society

www.OASUS.ca

Overview of SAS Enterprise Business Intelligence (3)

- Chapter 10: SAS Information Delivery Portal
  - Add content to pages
  - Add and edit portlets
  - Add and customize pages
  - Add a web report
- Chapter 11: Information Maps and OLAP Cubes
  - Create information maps using the Studio and PROC INFOMAPS
  - Create OLAP cubes using the Studio and PROC OLAP
  - Create an information map based on an OLAP cube

Ottawa Area SAS Users Society

www.OASUS.ca

Overview of SAS Enterprise Business Intelligence (4)

- Chapter 12: Web Report Studio
  - Use the Report wizard to create reports, tables, and charts
    - on datasets, information maps, and cubes
  - Customizing reports
  - Saving reports
- Chapter 13: SAS Stored Processes and the Stored Process Web Application
  - Running SAS code on the Web
  - Create and run a stored process in Enterprise Guide
  - Run a stored process using the Stored Process Web App
  - Run a stored process using the Web Report Studio

Ottawa Area SAS Users Society

www.OASUS.ca

How to develop SAS applications for the Web

- Chapter 3: Creating Static HTML Output
- Chapter 4: SAS and XML
- Chapter 8: SAS/IntrNet: htmSQL
- Chapter 7: SAS/IntrNet: the Application Dispatcher
- Chapter 14: Java Servelets and JavaServer Pages
- Chapter 15: SAS AppDev Studio 3.4 Eclipse Plug-in
- Chapter 16: SAS BI Web Services
- Chapter 17: Dynamic HTML with SAS and AJAX

Ottawa Area SAS Users Society

www.OASUS.ca

Chapter 3: Creating Static HTML Output

- Good old formchar "|----|+|---+=|-/\<>*"
- Writing HTML with PUT statements
- ODS HTML destination
  - Create multiple pages
  - Create a table of contents
  - Create HTML output from a DATA step
- Improving appearance with Style Sheets

and other exotic solutions in order to impose their own formats on output, or to select specific portions of the results for further processing.

One of the prime directives of object-oriented software development is **Thou shall not mix Data with Presentation**. Starting with SAS 7, the Output Delivery System was introduced to manage all the SAS procedure output in a consistent way. One or more *output objects* are created by each DATA step or procedure; these output objects each contain two basic components (Olinger 2000). The *data* component includes the raw data values that make up the results of the procedure or the contents of the Program Data Vector (PDV), while the *table* template (also called a table definition) describes how the data should be formatted. SAS has supplied a number of standard templates. You can modify these and even create your own using PROC TEMPLATE. C++ users should note that SAS uses the term "standard template" in a way that is quite different from what they may be used to. For more information about working with SAS templates, see the *PROC TEMPLATE FAQ and Concepts* at http://support.sas.com/rnd/base/topics/templateFAQ/Template.html.

As the online SAS System Help points out:

> ODS removes responsibility for formatting output from individual procedures and from the DATA step. The procedure or DATA step supplies raw data and the name of the table definition that contains the formatting instructions, and ODS formats the output. Because formatting is now centralized in ODS, the addition of a new ODS destination does not affect any procedures or the DATA step. As future destinations are added to ODS, they will automatically become available to all procedures that support ODS and to the DATA step.

The **Create HTML** check box selection described previously in fact just uses ODS. The drop-down menu shows you the list of the production style templates stored in the SASHELP.TMPLMST item store, so that you can pick the presentation style of the resulting output file. You can easily create your own HTML in batch mode using a few simple ODS statements that will accomplish the same effect. Example 3.4 shows how to do it; the HTML output and page source code are identical to that shown in Display 3.3 and Display 3.4.

Example 3.4 Using the ODS HTML Statement

```
filename OUT "Example 3-4.html";

ods listing close;
ods html body=OUT;

proc tabulate data=SASHELP.RETAIL;
   title "Retail Sales In Millions of $";
   class YEAR/descending;
   var SALES;
   table YEAR="" all="Total", SALES="" *
   (sum="Total Sales"*f=dollar8.
   pctsum="Overall Percent"*f=8.2
   n="Number of Sales"*f=8.
   mean="Average Sale"*f=dollar8.2
   min="Smallest Sale"*f=dollar8.
   max="Largest Sale"*f=dollar8.)/
   box="Year" rts=8;
run;

ods html close;
ods listing;
```

Example 3.6 Creating Frames with ODS

```
ods listing close;
ods html path="c:\Data\examples" (url=NONE)
        body = "body3-6.html"
        contents = "contents3-6.html"
        frame = "frame3-6.html";

***** Step #1 ****;
proc print data=SASHELP.RETAIL;
   title "1994 Sales Total by Month";
   where YEAR gt 1990;
   var MONTH SALES;
   id YEAR;
run;

***** Step #2 ****;
proc means data=SASHELP.RETAIL
        n mean min max
        nonobs fw=8 maxdec=2;
        title 'Retail Sales In Millions Of $';
        class YEAR/descending;
        var SALES;
run;

***** Step #3 ****;
proc tabulate data=SASHELP.RETAIL;

   class YEAR/descending;
   var SALES;
   table YEAR='' all='Total', SALES='' *
        {sum='Total Sales'*f=dollar8.
         pctsum='Overall Percent'*f=8.2
         n='Number of Sales'*f=8.
         mean='Average Sale'*f=dollar8.2
         min='Smallest Sale'*f=dollar8.
         max='Largest Sale'*f=dollar8.}/
   box='Year' rts=8;
run;

ods html close;
ods listing;
```

Opening the frame URL displays all three pages, as illustrated in Display 3.6. Clicking on a link in the left-hand window brings up the corresponding portion of the body. In addition to providing an absolute URL, it is also possible to specify `URL="NONE"`; the resulting HTML will include an anchor tag with a relative URL of the form `<a href="filename.ext">`.

This example shows a very simplistic page style. By using some simple SAS code, it is possible to format the table of contents so that it will have almost any desired appearance. There are many user-written papers about this topic; see the References section at the end of this chapter for more information.

Chapter 4: SAS and XML

- XML LIBNAME engine
- XML Mapper
- Creating XML with ODS markup
- Creating XHTML
- PROC TEMPLATE
  - Define tagsets, events, custom output
  - “PROC TEMPLATE listings can be intimidating, and many otherwise brave SAS users have avoided this procedure because of its reputation for complexity”

PROC TEMPLATE: Not Just for Geeks Anymore

As noted previously, the item store contains templates for tables and styles. It also contains the tagsets used for the ODS MARKUP statement. SAS supplies a simple XHTML template, or you can create your own by running the following program:

Example 4.11 XHTML Template

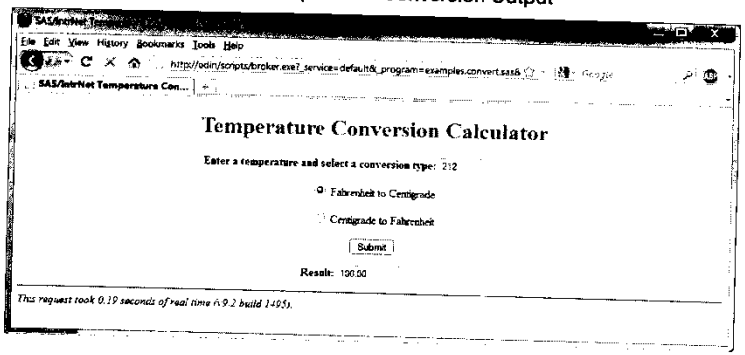
```
proc template;

define tagset Tagsets.xhtml / store = SASUSER.TEMPLAT;
   define event cell_is_empty;
   put %nrstr(" ");
end;

define event doc;
   start:
        put '<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"' NL;
        put '"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">' NL;
        put '<html xmlns="http://www.w3.org/1999/xhtml">' NL;
        ndent;
   finish:
        xdent;
        put "</html>" NL;
end;

define event doc_head;
   start:
        put '<head>' NL;
        ndent;
        put '<meta http-equiv="Content-Type"
        content="text/html;charset=utf-8" />' NL;
   finish:
        xdent;
        put "</head>" NL;
end;

define event doc_body;
   start:
        put "<body>" NL;
        put TITLE;
   finish:
        /* add W3C logo to page */
        put '<p>' NL;
        put '<a href="http://validator.w3.org/check/referer">' NL;
        put '<img src="http://www.w3.org/Icons/valid-xhtml11"' NL;
        put 'alt="Valid XHTML 1.1!" height="31" width="88" /></a>' NL;
        put '</p>' NL;
        put '</body>' NL;
end;

define event doc_title;
   put "<title>";
   put "SAS Output" / if !exists(VALUE);
   put VALUE;
   put "</title>" NL;
end;
```

Chapter 7: SAS/IntrNet: the Application Dispatcher

- Create the resources needed to use SAS code to dynamically generate HTML pages to be returned to a browser
- Generate dynamic output with ODS

Usually one needs to send more parameters to the server than just the name of the program. The Application Dispatcher simply translates all the name/value pairs into macro variables that are passed to the SAS program. In fact, we can create an interactive temperature conversion program easily in SAS as shown in **Display 7.16** and **Example 7.5**.

Display 7.16 SAS/IntrNet: Temperature Conversion Output



As noted in Chapter 3, the input text values are passed as parameters called `input` and `convert`. The SAS code saves these as macro variables `temp` and `type` respectively. The first time the page is loaded, these are empty. Once values have been inserted in the text boxes, pressing the `Submit` button on the form sends the values to the page, and the conversion code is executed, displaying the results. Note the two hidden fields on the page, which specify the values of `_service` and `_program`. This is the recommended way to pass these parameters.

Example 7.4 SAS/IntrNet: Temperature Conversion Program

```
/* SAS/IntrNet program to convert F to C and vice versa */

data _null_;

            file _webout;

            ***** write generic XHTML header *****;
            put '<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
                 Strict//EN"
            "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">'/
            '<html xmlns="http://www.w3.org/1999/xhtml" lang=
                 "en-US">';

            write top of page *****;
            put '<head>'/
            '<title>SAS/IntrNet Temperature Conversion
                Calculator</title>'/
            '</head>'/
            '<body>'/
            '<div style="text-align: center">'/
            '<h1 style="color: blue">Temperature Conversion
                Calculator</h1>';
```

```
**** create HTML form * with hidden text fields ***;
put '<form name="calculator" action="" method="get">'/
'<input type="hidden" name="_service" value="default"
    />'/
'<input type="hidden" name="_program" value="'
    "&_program" '"/>';

***** get parameter values *****;
temp = symget("input");
type = symget("convert");

***** input temperature *****;
put '<p><strong>Enter a temperature and select a
    conversion
    type: </strong>'/
'<input type="text" name="input" value="' temp '"/></p>';

***** select conversion/compute result *****;
put '<p><input type="radio" name="convert" value="1"' @;
if (type = '1') then do;
    put 'checked="checked"' @;
    result = 5*(input(temp,8.) - 32)/9;
end;
put '> Fahrenheit to Centigrade</p>';
put '<p><input type="radio" name="convert" value="2"' @;
if (type = '2') then do;
    put 'checked="checked"' @;
    result = 9*input(temp,8.)/5 + 32;
end;
put '> Centigrade to Fahrenheit</p>';

***** Submit button *****;
put '<p><input type="submit" value = "Submit"></p>';

***** Display results *****;
if (temp > ' ') then do;
    put '<p><strong>Result: </strong>'/
    '<input type="text" name="result" value="'
       result 6.2'"/> </p>';
end;

**** write bottom of page *****;
put '</form>'/
'</div>'/
'</body>'/
'</html>';
run;
```

The important parts of the program are shown in bold. There are five name/value pairs in this program: Two hidden input fields are used to encode the `_service` and `_program` parameters. These are passed to the broker to identify the default service and the temperature conversion program. The `&_program` macro variable contains the name of the program: `convert.sas`. The `input` and `convert` fields are used to collect the user input and pass it to the SAS program. These are converted from SAS macro variables into DATA step variables by the `symget` functions. Finally, the `result` field is used to display the answer; this field is also passed as a parameter that is never read by the SAS program.

Chapter 8: SAS/IntrNet: htmSQL

- htmSQL allows you to use SQL query syntax in IntrNet code
- You can use HTML forms to get the parameters for the SQL request

Example 8.1 Summarizing Data with htmSQL

```
<html>
<head>
   <title>SAS IntrNet Examples: htmSQL</title>
   <style type="text/css">
         body          { text-align: center; }
         caption { font-weight: bold; }
         h1      { color: blue; }
         h3      { color: red; }
         td      { text-align: right; }
         p.foot  { font-weight: bold; }
   </style>
</head>
<body>
{query server="odin:5010" sapw="user"
userid="sas" password="{sas002}65CC6A1B3B6EF24B409005161FBDA6C7"}
{sql}
   select product,
          sum(sales) as total label="Total Sales"
format=dollar8.
   from sashelp.shoes
   where region='{&region}'
   group by product
{/sql}
<h1>Shoe Sales by Product</h1>
<hr/>
{norows}
   <h3>No rows selected. Check that the region parameter
   has been specified correctly.</h3>
{/norows}
<table border="1" cellpadding="4" cellspacing="0"
       align="center">
       <caption>Region: {&region}</caption>
       <tr>
            {label var="{&sys.colname[*]}"
                        before="<th>"
                        between="</th><th>"
                        after="</th>"}
   </tr>
   {eachrow}
         <tr>
         { &{&sys.colname[*]}
               before="<td>"
               between="</td><td>"
               after="</td>" }
         </tr>
   {/eachrow}
</table>
```

Ottawa Area SAS Users Society

www.OASUS.ca

Chapter 14: Java Servelets and JavaServer Pages

- JavaServer Pages and Servlets
- JavaBeans
- JDBC
- An example with SAS

There are five standard tag libraries available:

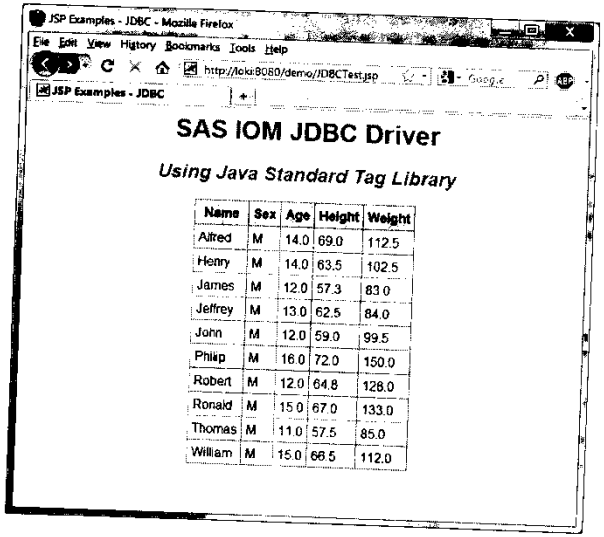
Table 14.2 Java Standard Tag Libraries

Functional Area	URI	Prefix	Example
Core	http://java.sun.com/jsp/jstl/core	c	<c:*tagname* ...>
XML processing	http://java.sun.com/jsp/jstl/xml	x	<x:*tagname* ...>
I18N capable formatting	http://java.sun.com/jsp/jstl/fmt	fmt	<fmt:*tagname* ...>
Database access	http://java.sun.com/jsp/jstl/sql	sql	<sql:*tagname* ...>
Functions	http://java.sun.com/jsp/jstl/functions	fn	fn:functionName(...)

The JSTL tags are designed for prototyping and not for production use. For industrial-strength applications, Oracle recommends encapsulating the functionality in JavaBeans components. Nonetheless, it is instructive to look at an example of some of the SQL tags available in the standard library.

The Web page shown in Display 14.9 can be constructed using JSTL with a minimum of Java code. The example displays the familiar SASHELP.CLASS data set as an HTML table. Note that this table is not editable, but it is reloaded every time the page is opened.

Display 14.9 JSTL Example



The code for this example is shown in Example 14.12. The first two lines reference the two required tag libraries from the Apache Jakarta project: the *core* library tags are referenced with the prefix "c" while the sql tag library uses the prefix "sql".

Example 14.12 Using JDBC with JSTL Custom Tags

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
<html>
<head>
   <title>JSP Examples - JDBC</title>
   <meta http-equiv="Content-Type" content="text/html; charset=utf-8"
   />
       <link href="styles.css" type="text/css" rel="stylesheet" />
</head>
<body>
       <h1>SAS IOM JDBC Driver</h1>
       <h2>Using Java Standard Tag Library</h2>
       <table border="1" align="center">
            <%-- open a database connection --%>
            <sql:setDataSource
              var="datasource"
              driver="com.sas.rio.MVADriver"
              url="jdbc:sasiom://localhost:8591" user="sas"
              password="{sas002}ACFD2406354EB53359FB76435B5FB53C"
  />
      <%-- execute the database query --%>
      <sql:query var="table" dataSource="${datasource}" >
           select * from sashelp.class where sex='M'
      </sql:query>

   <%-- Get the column names for the header of the table --%>
   <c:forEach var="columnName" items="${table.columnNames}">
      <th><c:out value="${columnName}"/></th>
   </c:forEach>

   <%-- Get value of each column by iterating over rows --%>
   <c:forEach var="row" items="${table.rowsByIndex}">
     <tr>
       <c:forEach var="column" items="${row}">
         <td><c:out value="${column}"/></td>
       </c:forEach>
     </tr>
   </c:forEach>
     </table>
</body>
</html>
```

The example program uses four JSTL tags to encapsulate the JDBC database actions described previously. The rest of the program is just standard HTML. An external style sheet is used to format the table. The four tags are as follows:

- sql:setDataSource – creates a connection to the JDBC data source. As the Jakarta documentation warns, you should not use this method for production Web sites. The details of using JINDI to manage connections are beyond the scope of this discussion.
- sql:query – passes the SQL select statement through to the SAS server. In this case, 20 records are selected.

Ottawa Area SAS Users Society

www.OASUS.ca

Chapter 15: SAS AppDev Studio 3.4 Eclipse Plug-in

- Shows how to use the AppDev Studio plug in for the Eclipse open source IDE to create Web applications.
  - “Do not attempt to use these tools unless you are already familiar with the Eclipse IDE, or are willing to spend some time learning the interface.“

Chapter 16: SAS BI Web Services

- Sending and receiving requests using PROC SOAP to consume a web service
- Creating a Web Service
  - Uses Stored Processes
- SAS provides an example

Example 16.1 SOAP Request

```
<soap:Envelope
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <GetCityForecastByZIP xmlns="http://ws.cdyne.com/WeatherWS/">
      <ZIP>27511</ZIP>
     </GetCityForecastByZIP>
   </soap:Body>
</soap:Envelope>
```

The exact format of the request depends on the server. Later in this chapter, we shall introduce Web Service Definition Language (WSDL) files, which specify the format for the request. CDYNE includes a listing of their WSDL for this operation on their site, but you do not need it at this point. For this request, note that the request is a well-formed XML document: the root element is `soap:Envelope`, while the actual operation is encoded in the `GetCityForecastByZIP` tag. Remember that XML is case-sensitive, so when creating a request, be sure to specify the formatting correctly.

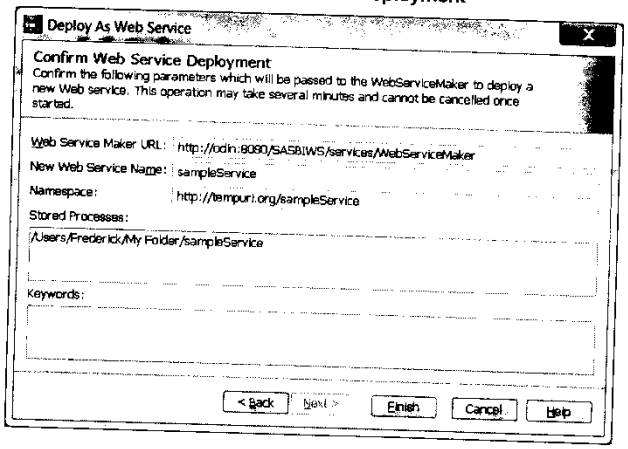
Incidentally, if you have a WSDL file but are not sure how to code the request object, there is a utility program called *soapUI*, from *eviware* (see http://www.soapui.org/ for a free trial download), which can read the WSDL and generate sample template requests.

When the request file has been created and saved to some convenient directory, the following SAS program will send the request to the service and then parse the retuned XML response.

Example 16.2 Sending a SOAP request

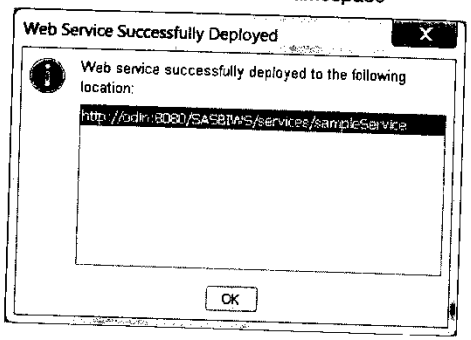
```
/*
 *  Path to SOAP messages
 */
filename REQUEST  "c:\data\examples\getWeatherRequest.xml";
filename RESPONSE "c:\data\examples\getWeatherResponse.xml";

/*
 *  Run SOAP request
 */
proc soap in=REQUEST
      out=RESPONSE
      url="http://ws.cdyne.com/WeatherWS/Weather.asmx"
soapaction="http://ws.cdyne.com/WeatherWS/GetCityForecastByZIP";
run;

/*
 *    Parse response data
 */
filename SXLEMAP 'c:\data\examples\getWeatherByZip.map';
libname  RESPONSE xml xmlmap=SXLEMAP access=READONLY;

data _null_;
   set RESPONSE.GetCityForecastByZIPResult;
   call symput("City",trim(city));
   call symput("State",trim(state));
   call symput("Station",trim(WeatherStationCity));
run;
```

Display 16.12 Confirm Web Service Deployment



Click **Finish** to create the service. If all goes well, you should see the URI for the new Web Service.

Display 16.13 Web Service Namespace



To make sure that the stored process has been registered in the metadata, you might want to choose the **Plug-ins** tab in SAS Management Console and select your active server under the **Metadata Manager**. Right-click and choose **Upgrade Metadata**. This will ensure that the service has been registered.

Deleting a Web Service

If you need to replace a SAS BI Web Service, SAS recommends that you delete the old one first. To do this in SAS Management Console, open the **\System\Services** folder and right-click on the

Ottawa Area SAS Users Society

www.OASUS.ca

Chapter 17: Dynamic HTML with SAS and AJAX

- "The development and maintenance of these applications is extremely difficult."
- HTML, JavaScript, and SAS/IntrNet
- JavaScript, JSON, and the SAS Stored Process Web Application
- JavaScript, XML, and SAS BI Web Services

Conclusions

- This is a very useful / essential book for anyone who wants to do web development with SAS.
- Anybody who wants to have an overview of web technologies should browse this book.
- Web technologies are still very kludgy.