**ABSTRACT**

Proper ingredients, precise measurements, and personal attention all contribute significantly to the creation of a perfect meal – the same can be said about implementing an effective software application. By knowing some fundamental SAS, HTML, and JavaScript elements and applying the right touch of each medium, one can easily develop practical and insightful programs for managing business systems. More specifically, by making use of these three languages, one can design valuable web-based solutions that exploit a number of technologies the Internet has to offer.

This paper will explain how basic components of SAS, HTML, and JavaScript can be fused together for implementing rather useful applications that function via the Internet. A background discussion on the SAS/IntrNet software will shed light on the advantages of integrating Internet technologies for realizing business solutions. Subsequently, a case study will focus on how these concepts were recently drawn together in a particular application deployed by the Data Processing Team of the Continuous Measurement Office at the United States Census Bureau. Finally, some information will be provided to offer insight on other programming tools that could be used together with SAS to create more extensive and comprehensive web-based applications, such as JavaServer Pages (JSP) and Servlets.

**BACKGROUND**

- **HTML**

  Webpages are written in a basic scripting language: HTML, or HyperText Markup Language. Essentially, HTML is a means of specifying layout information within documents. It is important to mention that HTML is not a programming language; rather, the markup grammar dictates the contents of an HTML file with no connection to instruction processing. In effect, a web browser renders an HTML document by seeking out distinctive HTML syntax used for altering the layout of the file, inserting images, and establishing links to other pages.

  There are 3 fundamental components to HTML: elements, tags, and attributes. The HTML directives, together with the text to which they apply, are called elements. A tag conveys the structure of the element to which it refers to rather than its appearance. Tags are denoted by enclosing < and > characters, and typically nest the elements. Start tags, those placed at the beginning of an element, may have attributes to define various characteristics of the contained elements, such as...
text alignment, format, or size. The following HTML code demonstrates all three of these components together:

```html
<H2 ALIGN="center"><I>An HTML Heading</I></H2>
```

The whole entity is an H2, or Heading, element, uses the start `<H2>` and `<I>` and closing `</H2>` and `</I>` tags, and utilizes the `ALIGN="center"` attribute. When rendered through a web browser, this statement would result in a large, italicized heading centered possibly at the top of a webpage, or perhaps above a subsection of text.

### JavaScript

JavaScript, a client-side scripting language executed by the user’s Internet browser, can conveniently be imbedded into HTML documents. Those designing webpages can benefit greatly from JavaScript because it can be exploited like a genuine object-based programming language. Given that JavaScript can be programmed to execute during or react to specific events, it is possible to dynamically change the content of an HTML element. Thus, JavaScript can be used to create responses to mouse clicks and keypress events, as well as being applicable for other practical functions, such as validating user entries before submitting a form to the web server. This can therefore help to reduce the overhead of server-side processing. The following scriptlet, inserted into HTML code, redirects the web browser to the U.S. Census Bureau’s homepage when the user clicks on the “Visit Census Site” button:

```html
<SCRIPT LANGUAGE="JavaScript"> function goToURL() { window.location = "http://www.census.gov"; }
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<input TYPE=button VALUE="Visit Census Site" onClick="goToURL()"
</FORM>
</BODY>
</HTML>
```

### SAS – ODS

The Output Delivery System, known as ODS, is a means for transforming SAS output into a variety of formats available to users, such as PDF, RTF, and HTML. By using the ODS HTML statement, one can create and store static HTML documents by denoting the destination path in the statement declaration:

```
ODS HTML HTML-file-specification(s) <option(s)>;
```

The HTML document generated is comprised of all the necessary HTML tags and attributes for it to be properly displayed within an Internet browser. These documents can be edited and later stored on a production web server so that users may view and evaluate these pages over the Internet.

What’s more significant is the ability to generate dynamic HTML content, which is where SAS/IntrNet software comes into play. Under the dynamic approach, a user request is sent from a web browser to a web server, which handles this request by invoking a SAS session. After processing the request, the program’s results are routed directly back to the web browser as HTML content, namely, a webpage. By creating webpages with this approach, Internet users retain control over what parameters are to be applied to the SAS program. Additionally, since these requests are all broadcasted via the web browser, users can send out requests merely by filling out uncomplicated web forms; thus, avoiding the need to write custom SAS code for programming each individual request.

### SAS/IntrNet

SAS/IntrNet is a valuable software tool used for processing dynamic SAS applications via the Internet. In particular, SAS/IntrNet enables communication between a web browser running on a local computer and a SAS session operating on a remote machine, namely, a web server.

The component most essential to the implementation of SAS/IntrNet is that of the application dispatcher. The application dispatcher, a program that runs on the same server where SAS and SAS/IntrNet are installed, governs the process of recognizing and responding to user requests relayed through an Internet browser. This process is quite straightforward and very effective in practice. To begin with, a user simply completes an HTML form using a web browser, such as Internet Explorer or Netscape Navigator. The entries recorded among the form’s fields will soon thereafter be used as parameters to an existing SAS program. The content of the form’s fields can depend on the complexity of the underlying SAS...
program, or simply on how much control the programmer wants to grant the user. The information obtained though the user input is then delivered to the web server, which in turn launches the application broker. The broker subsequently uses this information to determine which server should manage the request. At this time, the application broker passes the data to the SAS/IntrNet application server, another important piece to the dispatcher. Next, the application server invokes a SAS program, which ultimately processes the original information. The results are finally sent through the application broker and output back to the user’s browser as an HTML document—a webpage—for normal web viewing.

The application dispatcher is significant in that it provides the functionality of SAS to Internet users without accruing the overhead of installing SAS software on each client’s computer. Accordingly, a user simply needs a web browser to interact with and process SAS data; thus, users are not required to possess any SAS programming skills to be able to fashion colorful and constructive reports. In short, establishing a controlled and easily accessible system for managing data can readily be accomplished with this approach.

IMPLEMENTATION

• Problem

The Data Processing Team of the Continuous Measurement Office at the U.S. Census Bureau is responsible for processing the American Community Survey, or ACS. Traditionally, this group produces a variety of reports for subject-matter specialists to use when reviewing the edited ACS data. By and large, these specialists have been accustomed to sorting through extensive reports and substantial documentation in order to analyze data pertinent to their variables of interest.

A particular working example involves the analysts having to examine records relating to hot-decking matrix counts by state. However, there are so many records to consider, many of which are redundant or irrelevant, that viewing only the records that exist above a certain threshold would suffice.

• Solution

Implementing individual SAS programs to narrow down the sought-after data is an obvious solution; however, many of the analysts do not have enough, if any, SAS programming skills to achieve this. Furthermore, some analysts have too many requests—varying thresholds evaluated among different states—that any strategy for continually writing or modifying SAS code doesn’t seem viable. Therefore, for this instance and others alike, the employment of SAS/IntrNet software, along with common SAS and web development techniques, emerges as a logical and effective approach.

HTML

HTML can easily be applied to create Internet accessible forms, allowing users to select from a range of fields and/or enter in their own preferred values. After the user submits the form, the values are passed as parameters to an existing SAS program via the application dispatcher. This program is then executed on a remote server, which ultimately returns the results of the SAS code to the user’s web browser as an HTML document.

The <FORM> tag, along with some key attributes, is used for designing a form for maintaining the variables to be passed, as follows:

```html
<HTML>
<HEAD><TITLE>Form Example</TITLE></HEAD>
<BODY>
<H1 ALIGN=center>Please select a state. Choose Alabama, Alaska, or New Mexico</H1><HR><BR>
<form NAME=check METHOD=get ACTION="/../sas/scripts/broker.exe">
  <INPUT TYPE=hidden NAME=_service VALUE="dp">
  <INPUT TYPE=hidden NAME=_program VALUE="prgs.matrixcounts.sas">
  <INPUT TYPE=hidden NAME=thresh VALUE="0.5">
  <INPUT TYPE=hidden NAME=state VALUE="AL>

<SELECT NAME=state>
  <OPTION VALUE="01">Alabama
  <OPTION VALUE="02">Alaska
  <OPTION VALUE="56">Wyoming
</SELECT><BR><BR>
<SELECT NAME=thresh VALUE="0.5">
  <OPTION VALUE="0.5">0.5
  <OPTION VALUE="1.0">1.0
  <OPTION VALUE="2.0">2.0
</SELECT><BR><BR>
<SELECT NAME=state VALUE="01">
  <OPTION VALUE="01">Alabama
  <OPTION VALUE="02">Alaska
  <OPTION VALUE="56">Wyoming
</SELECT><BR><BR>
<INPUT TYPE=submit VALUE="Submit">
</form>
</BODY>
</HTML>
```

The Method= attribute used within the <FORM> tag defines the mode for passing the parameters, either get or post. The Action= attribute specifies the location of the application dispatcher program to be invoked somewhere on the web server. In general, the Name= attribute designates a unique variable name for a particular HTML element. A selection box and a textbox were utilized in this form, granting the user the ability to choose a particular state and enter a specific threshold. Other types of HTML data entry fields, such as textareas, multiple selection boxes, checkboxes, and radio buttons are also available. After making these selections, the user can submit the form using...
the corresponding button. Following submission, the application dispatcher launches the methodical process discussed in the SAS/IntrNet section above. In this particular case, there are 5 parameters delivered through the application dispatcher: 2 user-defined parameters and 3 hidden parameters defined by the programmer. The two parameters specified by the user are the chosen state code, state, and the threshold value entered, thresh. These will subsequently be used as macro variables in the SAS program, and referred to as &state and &thresh. The other three parameters, hard-coded into the HTML form, are central to the dispatcher process. The _program parameter determines the SAS program to be executed – in this instance, matrixcounts.sas. This program is located in the directory associated with the fileref prgs, which was defined on the SAS server beforehand. The parameter _service indicates the specific service assigned with the application dispatcher; generally, the value default can be used, but in this case, dp is employed since it has been prepared specifically by the system administrators. The final parameter, _debug, denotes the debugging mode exercised during the testing phase. For the most part, the SAS log is output to the web browser for error-checking and evaluation.

---

**JavaScript**

With the aid of JavaScript, webpages can be programmed to respond to various user events. In this particular case, JavaScript can be used to validate the user threshold entry prior to submitting the form to the web server. This can help reduce the overhead of server-side processing. To grasp this, consider the case where the user enters too small a threshold: in this situation, too many observations would be encountered in the data set, leading to a great deal of processing inside the SAS session. Consequently, the system may timeout due to a large execution time, or the browser may crash due to an overabundance of output. Also, bear in mind the case where the user enters an invalid numeric value. In this occurrence, the SAS program wouldn’t compile, yielding abnormal output, or possibly nothing at all. Hence, invoking a simple scriptlet to verify the threshold entry before the form submission would not only prevent unwarranted server-side processing, but would also circumvent the transmission of inadequate data.

```html
<HTML>
<TITLE>Form Example</TITLE>
<SCRIPT LANGUAGE="JavaScript1.2">
function checkNum() {
var x=document.check.thresh.value
var anum=~/^-?[0-9]*(\.[0-9]+)?$/
if (x=="" || anum.test(x)==false || x<0.5 || x>2.0) {
alert("Please enter a valid threshold between 0.5 and 2.0, or use the default value of 0.5.")
document.check.thresh.value = "0.5"
}
</SCRIPT>
</HEAD>
<BODY>
<H1 ALIGN="center"><FONT COLOR="#000000">
Matrix Counts Analysis</FONT></H1><HR><BR>
<FORM NAME=check METHOD=get ACTION="/../sas/scripts/broker.exe"
onSubmit="return false"
><FONT FACE="Comic Sans MS" SIZE="3">
1.) Select a State: <SELECT NAME=state>
</SELECT><BR><BR>
2.) Enter a Threshold <INPUT VALUE="0.5" SIZE="4" MAXLENGTH="4"
NAME=thresh onBlur="checkNum()">
<BR><BR>
3.) Submit Query: <INPUT TYPE=submit VALUE="Submit"
onClick="document.check.submit()">
<br><br>
<INPUT TYPE=hidden NAME=_program VALUE="prgs.matrixcounts.sas">
<INPUT TYPE=hidden NAME=_service VALUE="dp">
<INPUT TYPE=hidden NAME=_debug VALUE="2">
</FORM>
</BODY>
</HTML>
```

The added JavaScript here forces the user to enter a valid threshold between 0.5 and 2.0. After the user enters a value in the threshold textbox, the checkNum() function is called either if the cursor moves outside the textbox, or the user clicks somewhere else on the page (onBlur="checkNum()"). There is a unique case which this checkNum() function is not able to handle, namely, the case where a number is entered into the threshold textbox and the user then presses the ENTER key to submit the form. The onBlur event handler does not recognize any changes to the page, so the checkNum() function is never invoked. The form is then submitted even though the threshold entry may be invalid or vacant. Thus, the two other JavaScript pieces onClick="document.check.submit()" and onSubmit="return false" can be added to prevent the user from submitting the form with the ENTER key. The form can now be submitted with the mouse only after it has been properly completed. In brief, with...
the minor addition of these simple JavaScript elements, any prospect of passing an invalid or unusable parameter to the SAS program is avoided.

\[\text{SAS – ODS}\]

HTML and JavaScript are used for designing the front-end workings of these Internet applications; however, the most significant ingredient for creating these resourceful and practical web-based applications is the back-end program used for processing the data. SAS programs can use a parameter passed from an HTML form as a macro variable. For the current example, the user-selected state and keyed-in threshold value will be used as macro variables, as follows:

\[
\%\text{global file} ; \\
\%\text{let file = lib.state &state ;} \\
\text{libname lib '2001/adp5/edit_web/mtxdata/';} \\
\text{data gpratbad getnoput ;} \\
\text{set &file ;} \\
\text{if getvalue = . then getvalue = 0 ;} \\
\text{if putvalue = . then putvalue = 0 ;} \\
\text{if putvalue > 0 then do ;} \\
\text{gpratio = getvalue / putvalue ;} \\
\text{format gpratio 4.2 ;} \\
\text{if gpratio > &thresh and abs(getvalue - putvalue) > 5 then output gpratbad ;} \\
\text{end ;} \\
\text{else if getvalue > 5 then output getnoput ;} \\
\text{run ;} \]

The parameters \textit{state} and \textit{thresh} are treated as macro variables in the fragment of code above. The data sets holding the pertinent calculations are stored as state<##>, where ## refers to the designated SAS state code. Therefore, the results of this execution will reflect only the state selected by the user. The threshold variable will help to narrow the margin of records generated by each execution of this SAS program.

By applying the functionality of the Output Delivery System, reports and tables generated by the SAS code can be displayed colorfully in the client’s web browser. It is always important that the output is recognizable to the clients using these applications; thus, by knowing the users’ needs ahead of time and understanding how they would like the tables to appear, the SAS code can be written accordingly.

\[
\text{ods html body = _webout style=statdoc ;} \\
\text{data _null_ ;} \\
\text{file _webout ;} \\
\text{abbrev = fnamel(&state) ;} \\
\text{put "<b><i><font color="#003399">"abbrev" -- 2001</font></i></b>" ;} \\
\text{run ;} \\
\text{proc print data=gpratbdx noobs label ;} \\
\text{title "<center>Get/Put Ratios & &thresh</center>" ;} \\
\text{label getvalue = 'gets' ;} \\
\text{putvalue = 'puts' ;} \\
\text{gpratio = 'ratio' ;} \\
\text{run ;} \\
\text{%let fn =<center><form><input type="button" value="Back" onClick="parent.location='javascript:history.back()'"> </form></center>;} \\
\text{proc print data=getnoptx noobs label ;} \\
\text{title "<center>Gets and No Puts</center>" ;} \\
\text{label getvalue = 'gets' ;} \\
\text{putvalue = 'puts' ;} \\
\text{gpratio = 'ratio' ;} \\
\text{footnote &fn ;} \\
\text{run ;} \\
\text{ods html close ;} \]

The above code produces the HTML reports that are to be displayed in the user’s web browser. It is important to note that the output destination _webout is used to target the results of this execution directly to the web browser making the original request. Contrary to this dynamic approach, an actual path location can be written in place of _webout. Then, after executing the program, the resulting HTML documents can be stored in the appropriate location on the web server. In this fashion, the programmer can hard-code hyperlinks pointing to these documents, so that any user can view their content directly by clicking on the particular link.

It is also apparent that HTML and JavaScript can be embedded directly into the SAS code giving the user more control over the ODS output. In the above example, a title and footer will be added to the resulting HTML document. The title will include the user’s state selection and the footer will contain a button that links to the previous webpage.

The option \textit{STYLE=} applies predefined SAS styles, such as Brick, Beige, or D3D, to the output HTML documents. These can be applied to enhance the appearance of a webpage; however, the use of a style will not affect the actual content of that page. The SAS procedure \textit{PROC TEMPLATE} can be used to craft new styles, which may be more applicable to the programmer’s taste, or the project’s requirements. Another practical ODS HTML option is \textit{STYLESHEET=} which the programmer can use to create new or apply existing \textit{Cascading Style Sheets}. Cascading Style Sheets, or CSS, define a unique layout for a web document by describing
how HTML elements should be displayed. Furthermore, CSS allow developers to control the style and layout of multiple webpages all at one time; so, making a global change to a number of documents can be achieved through a single modification. Other ODS HTML options give way to the construction of navigational menus and frames, namely, the BODY=, CONTENTS=, PATH=, PAGE=, and FRAME= options.

**CONCLUSION**

By applying an appropriate measure of SAS and other programming components, it is possible to design a controlled system for managing data that is accessible via the Internet. To begin with, developers can craft webpages using HTML (and possibly JavaScript) that contain forms to encapsulate user input. After a user completes and submits a given form, the request is then routed to a web server, which in turn launches a SAS session. Ultimately, the request is processed and the results are directed back to the client’s web browser as a colorful and comprehensive webpage. SAS/IntrNet essentially provides a channel for clients, perhaps those who lack programming skills, to interact with SAS data through their web browsers. There are other tools available which can be used to form more complex webpages that may include colorful graphs or navigational menus. It should be apparent that with the tools now available through SAS, coupled with the modern technologies of the Internet, providing clear and readable data to clients can, convincingly, be achieved.

**RESOURCES**

SAS® Institute, Inc. – Instructor-Based Training:
- SAS® Web Tools: Static and Dynamic Solutions Using SAS/IntrNet Software
- SAS® Web Tools: Advanced Dynamic Solutions Using SAS/IntrNet Software
- SAS® Web Tools: Developing JavaServer Pages and Servlets Using webAF™ Software

SAS® Institute, Inc. – Base SAS Community:

SAS and all other SAS Institute, Inc. product or service names are registered trademarks or trademarks of SAS Institute, Inc. in the USA and other countries. Other brand or product names are registered trademarks or trademarks of their respective companies.

**CONTACT INFORMATION**

Jonah P. Turner  
United States Census Bureau  
4700 Silver Hill Road, Mail Stop 8400  
Washington, DC 20233-8400  
(301) 763-5420 or jonah.p.turner@census.gov