Paper 025-29

A Recipe for Success: Migrating SAS/AF® Applications to the Web Using SAS®, HTML, and JavaScript

Jonah P. Turner, United States Bureau of the Census, Washington, D.C.

ABSTRACT

The Internet is a widespread information network that has reformed organizational structures and transformed business strategies. Given the predominance of cross-platform, global clientele and the ever-increasing need to deliver information, it serves great value for organizations to exploit web technologies. SAS programmers can adapt to these modern technologies by retooling their applications to provide clients with more accessible, user-friendly web-based solutions. Specifically, by drawing on various web programming techniques, a developer can seamlessly rework a SAS/AF program into an application that functions via the Internet.

As a follow-up to "A Pinch of SAS®, a Fraction of HTML, and a Touch of JavaScript Serve Up a Grand Recipe," published in the SUGI 28 proceedings, this paper will elaborate on the subject of web migration by discussing how programmers can go about modernizing their existing SAS/AF applications using SAS, HTML, and JavaScript while still preserving the underlying SCL code that make up these programs. Adapting these SAS/AF applications to the Internet allows organizations to maintain legacy programs and remain up to date with current trends in the ever-changing world of the Information Age.

INTRODUCTION

SAS/AF software gives programmers the ability to design interactive windowing applications that provide users fast and easy access to SAS data. Using the FRAME entry in SAS/AF, developers can create a graphical user interface (GUI) that has window elements such as icons, selection menus, command buttons, and scroll bars. Integral to the SAS/AF software is SCL, or SAS Component Language (formerly called Screen Control Language), a programming language designed to facilitate the development of these interactive applications. In effect, application developers can make the most of SCL by linking information obtained from programming statements with interactive input from users. In addition to its own programming statements, functions, and call routines, SCL provides a means for generating SAS source code for execution by the SAS System. SCL can also output other types of source code, such as HTML and JavaScript, for execution by a remote host. Collectively, these features give developers the ability to take SAS/AF programs and re-establish them as web-based applications.

The material contained herein is intended to help guide programmers in taking their existing SAS/AF applications and transforming them to function via the Internet. A brief discussion of SCL, along with some background information on HTML, JavaScript, and SAS ODS concepts, will provide insight into how each component plays a unique role in the development of web-based applications. For supplementary background information on the latter topics, it is recommended that the reader review the paper "A Pinch of SAS®, a Fraction of HTML, and a Touch of JavaScript Serve Up a Grand Recipe," published in the proceedings for SUGI 28, which serves as a precursor to this discussion. This paper will then demonstrate how each of these components was drawn together in the conversion of a particular SAS/AF application used by various personnel operating out of the National Processing Center for the United States Bureau of the Census. Any application developer looking to migrate existing SAS/AF programs to function through the Internet can undoubtedly benefit from this paper given the uncomplicated and straightforward nature of the discussed approach.

BACKGROUND

- ***SCL and FRAMES***

SAS Component Language is used to build SAS/AF (Application Facility) programs. In addition to its own programming facility, SCL provides the same basic elements and functionality as the base SAS language. Moreover, with the advent of FRAME entries, a GUI can be coupled with an SCL program to give users easy point-and-click navigation throughout the application. SCL variables associated with the fields of a FRAME entry provide a communication channel between SCL programs and end-users, that is, window fields are used to not only accept and pass user input through the underlying code, but also to display the resulting output of the program for users to observe. Many organizations have developed applications using SAS/AF software to capitalize on its windowing capabilities; however, by employing these applications through the Internet, organizations can further the utility and accessibility of these programs by way of extending to more cross-platform, global end-users.

The following example depicts a SAS/AF FRAME entry designed to accept a survey ID (i.e. a CMID) and display various status codes corresponding to the household associated with that form identification number:

Use SAS/AF Software to Build a FRAME Entry →

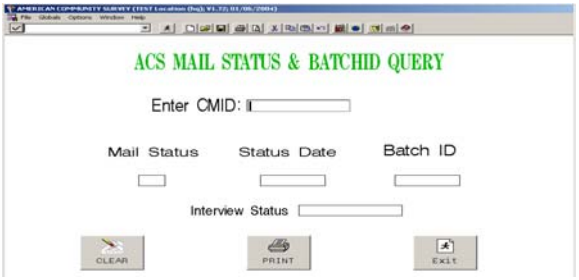

Figure 1. SAS/AF FRAME Entry Example

- ***HTML***

The HyperText Markup Language (HTML) is a grammar used to define layout information of a document. By embedding HTML syntax, it is possible to better control the structure and formatting information of a document rendered through a web browser, as well as add enhanced functionality to it. As such, developers can publish documents to the Internet in a platform-independent format, create links to related documents and other Internet information resources, and incorporate graphics and different types of multimedia within their documents.

The basic components of HTML, *tags* and *attributes*, together with the text to which they apply make up HTML *elements*. Whereas tags establish the structure of an element, attributes are used to define various characteristics, such as text alignment, font, and size. Start and end tags, identified by enclosing < and > characters, are placed around segments of text, with the end tag including a / character. Attributes can be placed within start tags to provide design information about the corresponding element.

The following example illustrates an anchor element, which is used to generate a hyperlink, where *<A>* and *</A>* are the start and end anchor tags, respectively, and *HREF* is an attribute that specifies the URL of a given webpage:

```
<HTML><TITLE>Link Example</TITLE>
  <BODY>
    <A HREF="http://www.google.com">Click to Visit Google</A>
  </BODY>
</HTML>
```

→

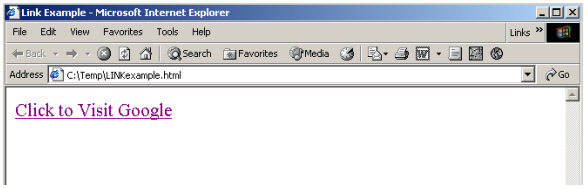

Figure 2. HTML Example

- ***JavaScript***

JavaScript is an object-oriented, platform-independent, interpreted scripting language that can be embedded into HTML documents to generate web content dynamically. In practice, JavaScript can enhance the level of interactivity and functionality of a webpage by way of event handling and client-side execution. Developing HTML documents with embedded JavaScript can give programmers more control over the end-user's web browser. Accordingly, JavaScript can be used to trigger a response to client-side activity, such as mouse clicks, keypress events, and page navigation. As follows, JavaScript is often programmed to validate user input into HTML form fields, which ultimately reduces the overhead of server-side processing.

The following example demonstrates how JavaScript can be embedded into an HTML document using the enclosing *<SCRIPT>* and *</SCRIPT>* tags in order to set off a dynamic response to a mouse click:

```
<HTML><TITLE>JavaScript Example</TITLE>
  <HEAD>
    <SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
      function okURL() {
        return confirm("Are you sure you want to leave this site?") ;
      }
    </SCRIPT>
  </HEAD>
  <BODY>
    <A HREF="http://www.google.com" onClick="return okURL();">
    Click to Visit Google</A>
  </BODY>
</HTML>
```

→

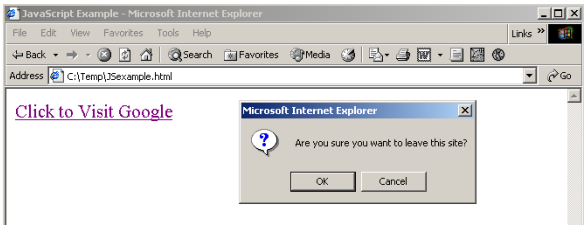

Figure 3. JavaScript Example

- ***SAS ODS***

The SAS Output Delivery System (ODS) provides a means for channeling SAS output into a variety of formats. This valuable component of the SAS System offers programmers added control for creating customized procedure output. One such format for deploying procedure output is HTML, where the results of an executed SAS program are transformed into a document that can be rendered through a web browser. Using the *ODS HTML* statement, programmers can explicitly define a destination path for storing procedure output in a web document:

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

SAS/IntrNet software facilitates a dynamic approach, in which these web documents can be generated on the fly as end-users send requests to the SAS System by simply filling out and submitting HTML forms. Under this approach, a user request is sent to a web server, which in turn invokes a SAS session. The results of this instance of the program are then redirected back to the user's web browser as a colorful and comprehensive webpage.

The following example shows how the procedure output of an executed SAS program using the ODS HTML statement (with embedded HTML and JavaScript syntax) can be displayed in a web browser:

```
data records ;
   input Team $ Wins Losses ;
   datalines ;
Packers 9 0
Vikings 6 3
Lions   4 5
Bears   2 7
;
run ;

ods html body = 'C:\Temp\ODSexample.html' ;
title 'NFC North Standings' ;
%let fn = '<INPUT TYPE=button VALUE=Close
          onClick="window.close();">' ;
proc print data = records noobs ;
   footnote &fn ;
run ;
ods html close ;
```

→

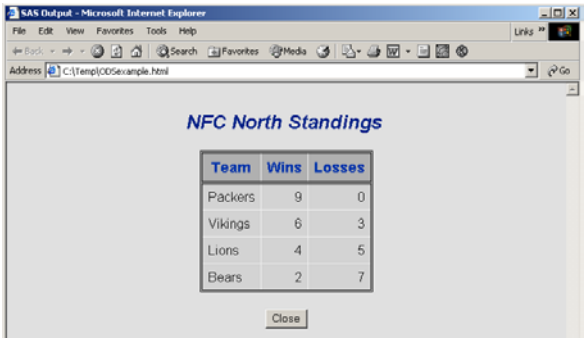

Figure 4. SAS ODS Example

IMPLEMENTATION

- ***Problem***

The support staff at the National Processing Center for the United States Bureau of the Census have a system in place to record, look up, modify, and examine the status and other select attributes of any given American Community Survey case. This particular instrument was developed using SAS/AF software given its unique windowing and multi-user design capabilities. Despite the success of this system in operation, employing it any further would require the overhead of having to install SAS on each terminal to be used. In addition, it can be expected that problems will arise when attempting to make this application accessible across different system platforms. Having a web-based version of this application in place would not only help to sort out these key issues, but also make it easier to maintain and enhance the system. Be that as it may, seeing as this application was designed using an extensive amount of SCL, and considering that it has functioned reliably in production for some time, there is great value in trying to maintain as much of the underlying code as possible.

- ***Solution***

The 2 major components that make up a typical SAS/AF application are the FRAME entry, used to accept and display user input, and the SCL entry, which holds the underlying code to validate and process this information. When converting to a web-based system, HTML and JavaScript will replace the FRAME component and be used to fashion a GUI that accepts and validates user input, as well as displays program output. The SAS ODS can also be exploited to render procedure output in a clear and easy manner. SCL code will be preserved and still be used to process user input and handle data set manipulation as it does in the SAS/AF environment.

With an integrative approach, it is possible to adapt this SAS/AF application to the Internet while still preserving much of the SCL code used in the programs currently in production. The example to be discussed is one of the many programs involved with this SAS/AF application. This particular program lets users enter in a CMID and perform a phone number lookup on the household associated with that form identification number (Figure 5). It should be noted that this particular program could conceivably be redesigned using other techniques; however, given its simple and uncomplicated makeup, it is a fitting example to effectively demonstrate the discussed approach.

SCL and the Web

It is very easy to embed HTML, JavaScript, and SAS code directly into an SCL program using the *SUBMIT* statement. The SUBMIT command labels the beginning of a block of programming statements to send to the SAS System for execution:

SUBMIT <*when* <*where*> <*host*>> <'STATUS'> ;

When a SUBMIT block is encountered, SCL gathers all the programming statements between the SUBMIT and ENDSUBMIT commands and places this content in the *PREVIEW* buffer. The PREVIEW buffer will retain the statements generated by the SUBMIT block before they are executed:

rc = **PREVIEW**(*action*<,*argument-1*<,*argument-2*<,*argument-3*>>>) ;

Specifically, by targeting the contents of the PREVIEW buffer to the file *_webout*, the statements within the corresponding SUBMIT block will be immediately redirected back to the web browser of the user making the request. Thus, programmers can strategically place blocks of HTML, JavaScript, and SAS code throughout their SCL programs in order to fully control the timing of when output should be displayed in the user's web browser.

Executing a compiled SCL program through the Internet is done in the same fashion as any other SAS/IntrNet application, namely, by invoking the following URL from a web browser:

http://*web.server.name*/*cgi-dir*/*broker*?_program=*program.name*&_service=*service.name*[&*varname1*=[*value*]&*varname2*=[*value*]…]

When developing a SAS/IntrNet application, it is important to remember that the web is a stateless environment, that is to say, a client request to the server is independent of all preceding requests. This can pose a burden on application developers as it is may be necessary to share information from one request to the next. For basic applications, this is not an issue because it is very easy to pass all the necessary variables to every new request by simply appending each one to the URL suffix. Nevertheless, developers often need to maintain items such as SAS data sets or SCL lists, which cannot easily be carried over to subsequent requests. In cases such as these, programmers have the ability to maintain the state by explicitly creating a SAS *session* and continue passing the unique session ID associated with it across multiple stored process requests. A session consists of macro variables and library members that the stored process has explicitly saved in the reserved *SAVE* library, and is scoped so that each user's session is separate from others. For the example described in this paper, however, it is not necessary to maintain a session since each CMID lookup is independent of all other requests. Essentially, a new instance of the SCL program is executed on every request with the unique user-entered CMID value always appended to the URL.

HTML

As previously discussed, HTML can be used to build a graphical interface that accepts user input and displays program output. Rather than having to build a separate FRAME entry using SAS/AF software, the programmer can now simply embed HTML directly into the SCL program using a SUBMIT block. As a general rule, when a SAS/AF application is run, execution begins in the *INIT* section – it is here where the programmer should insert the HTML statements that make up the GUI. Therefore, as program execution first jumps to the INIT block, the embedded HTML statements will be sent to the PREVIEW buffer, which points to the file *_webout*, and promptly rendered in the user's web browser. The interface for this application is comprised of an input field that accepts a CMID and assigns its value to the form variable *htmlid*, in addition to four disabled input boxes used to display phone numbers corresponding to the household associated with the user-entered CMID (Figure 6). The variable *htmlid* will be appended to the URL with each submission of the form and made available for the current execution of the SCL entry. According to the method discussed in the previous paper, any variable affixed to the URL is passed to the specified SAS program as a global macro variable. With this approach, values submitted to an SCL program are delivered in the form of an SCL list. Each variable must first be extracted from the SCL list and assigned to a unique SCL variable before ever being applied in other SCL statements or within the programming commands of any enclosed SUBMIT block by way of adjoining it with a preceding *&* character.

By default, this application is first invoked using only the variables *_program=prg.js.ph.scl* and *_service=acs* in the suffix of the URL. The application broker uses the values of *_program* and *_service* to determine which SCL program to invoke and with what service configurations. With the exception of the initial invocation, the CMID entered by the user will always be passed to the SCL program, extracted from the entry SCL list, and assigned to the SCL variable *sclid*. Since *htmlid* is not a parameter on the initial load, it will not appear in the SCL list and, thus, *sclid* is initialized to blank. Therefore, after the HTML statements that make up the GUI are channeled back to the user's web browser, program execution ceases as no lookup is performed (i.e. if sclid ne " " then link LOOKUP ;). The CMID input box will appear empty (i.e. <INPUT MAXLENGTH="9" NAME=htmlid VALUE=&sclid>) and wait for user input. When the user enters a CMID and submits the form, in effect invoking the same program, the variable *htmlid* is appended to the URL and subsequently passed on as a parameter to the SCL program. This time

the SCL list holds the user-entered CMID and, as a result, *sclid* is assigned a value. After the GUI is redisplayed, a lookup can be performed given that *sclid* is no longer blank. Finally, the CMID value populates the input field, along with any corresponding phone numbers that may have been found for that CMID during the table lookup.

Approach 1: SCL and FRAMES

Use SAS/AF Software to Build a FRAME Entry →

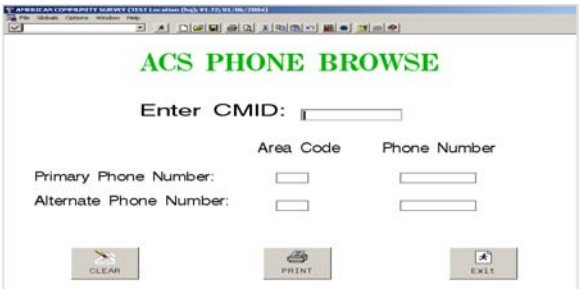

Figure 5. SAS/AF Version of GUI

Approach 2: SCL and the Web

```
/* Read all parameters into an SCL list */
entry optional list = 8 ;

INIT:

 /* Extract the user-entered cmid from the SCL list (if it exists) */
item = nameditem(list, 'htmlid') ;
if item ne 0 then sclid = getnitemc(list, 'htmlid') ;  else sclid = " " ;

/* The HTML code within this submit block will generate the GUI */
/* The initial load, sclid will be " " so the input box will be empty */
/* When the user enters a cmid and submits the form, the loaded */
/* page will display the cmid in the input box for the user to view */
submit;
<HTML><TITLE>Phone Browse</TITLE><BODY><CENTER>
<FONT COLOR="green" SIZE="+3"><B>ACS PHONE BROWSE</B>
</FONT><BR><BR><BR>

<FORM NAME=idform>
  <FONT SIZE="5"><B>Enter CMID: </B></FONT>
  <INPUT MAXLENGTH="9" NAME=htmlid VALUE=&sclid>
  <INPUT TYPE=hidden NAME=_program VALUE=prg.js.ph.scl>
  <INPUT TYPE=hidden NAME=_service VALUE=acs>
</FORM><BR>

<FORM NAME=phform><TABLE CELLPADDING="6">
  <TR><TD> </TD>
        <TD ALIGN="center"><B>Area Code</B></TD>
        <TD ALIGN="center"><B>Phone Number</B></TD></TR>
  <TR><TD>Primary Phone Number:</TD>
        <TD ALIGN="center">
        <INPUT SIZE="3" NAME=ptela DISABLED></TD>
        <TD ALIGN="left">
        <INPUT SIZE="7" NAME=pteln DISABLED></TD></TR>
  <TR><TD>Alternate Phone Number:</TD>
        <TD ALIGN="center">
        <INPUT SIZE="3" NAME=atela DISABLED></TD>
        <TD ALIGN="left">
        <INPUT SIZE="7" NAME=ateln DISABLED></TD></TR>
  <TR ALIGN="center">
        <TD><INPUT TYPE=reset VALUE=Clear></TD>
        <TD><INPUT TYPE=button VALUE=Print></TD>
        <TD><INPUT TYPE=button VALUE=Exit></TD></TR>
</TABLE></FORM>

</CENTER></BODY></HTML>
endsubmit ;
rc = preview('FILE', '_WEBOUT') ;
rc = preview('CLEAR') ;

/* If a cmid was entered by the user, then proceed to execute the */
/* SCL commands that look up the phone numbers for the cmid */
if sclid ne " " then link LOOKUP ;

return ;

LOOKUP:
  <Insert SCL to look up the cmid and display phone numbers>
return ;
```

→

Figure 6. Web Version of GUI

Comparing the screenshot of the web-based version (Figure 6) with that of the SAS/AF version (Figure 5), it is clear to see that with no more than HTML, it's easy to build a web interface that closely models a SAS/AF FRAME entry.

JavaScript

HTML has now been used to form a basic web user interface; though, it is still necessary to build in various JavaScript routines in order to model general SCL and FRAME functionality, validate user input for client-side processing, generate warning and error alert messages when needed, and ultimately display program results.

To enhance usability, the cursor should always be positioned in the CMID input field whenever the screen is loaded. Formerly, an SCL routine (i.e. call notify('cmid', '_cursor_') ;) was inserted at the beginning of the INIT block to handle this. For the web-based version, the JavaScript *focus()* function, used in conjunction with the *onLoad* event handler, emulates the same browser functionality (i.e. <BODY onLoad="document.idform.htmlid.focus();">).

JavaScript can also be used to add functionality to the HTML print and exit buttons that are integrated in the GUI:

Approach 1: SCL and FRAMES

```
/* Execution jumps here when the user clicks the print or exit buttons */
PRINT:
   call execcmdi('dlgprt nosource activebitmap', 'EXEC') ;
return ;
EXIT:
   call execcmd('END') ;
return ;
```

↔

Approach 2: SCL and the Web

```
<!-- JavaScript routines added to the HTML print and exit buttons -->
<TD>
  <INPUT TYPE=button VALUE=Print onClick="window.print();">
</TD>
<TD>
  <INPUT TYPE=button VALUE=Exit onClick="window.close();">
</TD>
```

The former approach for displaying warning and error messages involved opening other FRAME entries. The same can be achieved by inserting JavaScript functions within the HTML statements that build the interface. For instance, any illegal value entered in for the CMID, namely, anything that is not a 9-digit number, should trigger a warning:

Approach 1: SCL and FRAMES

```
/* Execution jumps here when the user enters in a cmid */
CMID:
   call notify('cmid', '_get_text_', sclid) ;
   if (verify(sclid, '0123456789') > 0 or length(sclid) ne 9) then do ;
      call wregion(7,33,23,50) ;
      call display('WARN.frame', 'WARNING!', 'Bad CMID
                  Entered', sclid, 'Please Re-Enter CMID') ;
      call notify('cmid', '_set_text_', ' ') ;
      call notify('cmid', '_cursor_') ;
      return ;
   end ;
   else link LOOKUP ;
return ;
LOOKUP:
   <Insert SCL to look up the cmid and display phone numbers>
return ;
```

→

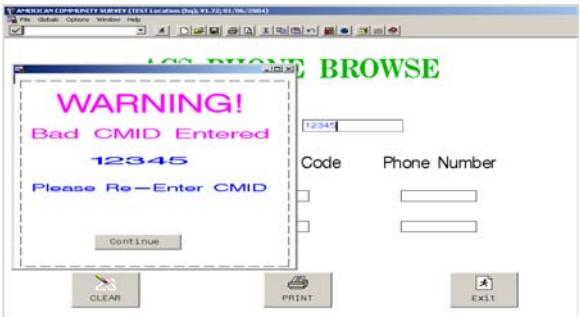

Figure 7. SAS/AF Version of Warning Message

Approach 2: SCL and the Web

```
<!-- Insert this JavaScript function to validate the user input -->
<HTML><TITLE>Phone Browse</TITLE>
  <HEAD><SCRIPT LANGUAGE="JavaScript1.2">
    function checkCMID() {
      var x = document.idform.htmlid.value ;
      var anum = /\d{9}/ ;
      if (anum.test(x) == false) {
        alert("     WARNING! \n\n Bad CMID Entered\n\n      "
            + x + "\n\nPlease Re-Enter CMID") ;
        document.idform.htmlid.value = '' ;
        document.idform.htmlid.focus() ;
      } else document.idform.submit() ;
    }
  </SCRIPT></HEAD>
</BODY><CENTER>
<FONT COLOR="green" SIZE="+3"><B>ACS PHONE BROWSE</B>
</FONT><BR><BR><BR>
<!-- Invoke the JavaScript function whenever the form is submitted -->
<FORM NAME=idform onSubmit="checkCMID(); return false;">
```

→

Figure 8. Web Version of Warning Message

To this point, a web interface that accepts and validates user input has been created by way of embedding HTML and JavaScript programming statements inside the SCL entry code. The first time this SCL program is invoked, the GUI is rendered in the user's web browser and waits for user input. After a CMID has been keyed in and validated, the form is submitted (by pressing the ENTER key), which in turn invokes the same SCL program; however, this time *sclid* has been set to the value entered by the user. As a result, program execution can jump to the LOOKUP

block to run the SCL that processes the CMID – this code remains more or less the same for both versions of the application with minor differences involving the ways in which errors, warnings, and program results are displayed:

Approach 1: SCL and FRAMES

```
LOOKUP:   /* Execution jumps here when a cmid has been entered */
   dsid = open('npc.phonebrs', 'i') ;    /* Open in browse mode */
   if dsid = 0 then do ;
      call wregion(7,33,23,50) ;
      call display('WARN.frame', 'WARNING!', 'NPC Open Failed',
                   '  ', 'Try Again Later') ;
      return ;
   end ;

   whcls = 'cmid = ' || quote(sclid) ;
   rc = where(dsid, whcls) ;
   rc = attrn(dsid, 'any') ;  /* If observations match where, rc=1 */
   if (rc ne 1) then do ;
      call wregion(7,33,23,50) ;
      call display('WARN.frame', 'WARNING!', 'No Records Found',
                   sclid, 'Enter a New CMID') ;
      call notify('cmid', '_set_text_', ' ') ;
      call notify('cmid', '_cursor_') ;
      rc = where(dsid, 'clear') ;
      return ;
   end ;
   rc = fetch(dsid) ;
   if (rc = 0) then do ;    /* Display results in phone number boxes */
      area1    = getvarn(dsid, varnum(dsid,"tela")) ;
      phone1 = getvarn(dsid, varnum(dsid,"teln")) ;
      area2    = getvarn(dsid, varnum(dsid,"atela")) ;
      phone2 = getvarn(dsid, varnum(dsid,"ateln")) ;
   end ;
return ;
```

→

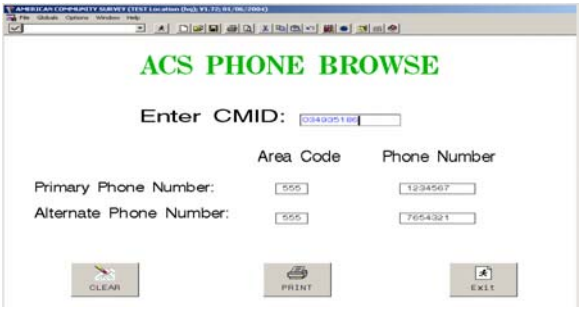

Figure 9. SAS/AF Version of Displayed Results

Approach 2: SCL and the Web

```
LOOKUP:   /* Execution jumps here when a cmid has been entered */
   dsid = open('npc.phonebrs', 'i') ;    /* Open in browse mode */
   if dsid = 0 then do ;
      submit ;
      <script>
        alert("WARNING! NPC Open Failed\n\nTry Again Later") ;
      </script>
      endsubmit ;
      rc = preview('FILE', '_WEBOUT') ;
      rc = preview('CLEAR') ;
      return ;
   end ;

   whcls = 'cmid = ' || quote(sclid) ;
   rc = where(dsid, whcls) ;
   rc = attrn(dsid, 'any') ;  /* If observations match where, rc=1 */
   if (rc ne 1) then do ;
      submit ;
      <script>
        alert("WARNING! No Records Found\nEnter a New CMID") ;
        document.idform.htmlid.value = '' ;
        document.idform.htmlid.focus() ;
      </script>
      endsubmit ;
      rc = preview('FILE', '_WEBOUT') ;
      rc = preview('CLEAR') ;
      rc = where(dsid, 'clear') ;
      return ;
   end ;
   rc = fetch(dsid) ;
   if (rc = 0) then do ;    /* Display results in phone number boxes */
      area1    = getvarn(dsid, varnum(dsid,"tela")) ;
      phone1  = getvarn(dsid, varnum(dsid,"teln")) ;
      area2    = getvarn(dsid, varnum(dsid,"atela")) ;
      phone2  = getvarn(dsid, varnum(dsid,"ateln")) ;
      submit ;
      <script>
        document.phform.ptela.value = &area1 ;
        document.phform.pteln.value = &phone1 ;
        document.phform.atela.value = &area2 ;
        document.phform.ateln.value = &phone2 ;
      </script>
      endsubmit ;
      rc = preview('FILE', '_WEBOUT') ;
      rc = preview('CLEAR') ;
   end ;
return ;
```

→

Figure 10. Web Version of Displayed Results

SAS ODS

As previously discussed, it is possible to embed SAS language inside an SCL program for execution by the SAS System using SUBMIT blocks. This can serve great value for developers who are more proficient or comfortable with DATA step programming and, therefore, may facilitate the process of migrating SAS/AF applications to the Internet.

One major benefit of involving the SAS ODS in the migration process is that SAS procedure output can be controlled and enhanced to offer users more illustrative, rich content that is readable in a web browser. For this application, the results of the executed SCL program can be displayed in the client's web browser using ODS HTML, rather than by means of dynamically populating HTML disabled input fields using JavaScript. This may be a more sensible approach to follow when converting SAS/AF applications that require a great deal of displayed output, in which it suffices to make use of built-in SAS procedures.

Whereas it is necessary to explicitly redirect the contents of the PREVIEW buffer in order to render the embedded HTML and JavaScript syntax in the client's web browser, programmers can instead conveniently specify the *_webout* destination in the *BODY* option of the ODS HTML statement to readily deliver procedure output to the end-user (i.e. ods html body = _webout style = statdoc ;).

The following snippet demonstrates an alternative method for displaying program output for the web-based version of the discussed application – in this case, the phone number results are presented in a rich, colorful chart rather than in standard HTML input fields:

Approach 2 (Alternative): SCL and the Web

```
/* Remove all HTML phone number disabled input boxes from GUI */
/* Now, the results will be displayed using SAS ODS HTML instead */
LOOKUP:   /* Execution jumps here when a cmid is entered */
   dsid = open('npc.phonebrs', 'i') ;    /* Open in browse mode */
   if dsid = 0 then do ;
      submit ;
      <script>
         alert("WARNING! NPC Open Failed\n\nTry Again Later") ;
      </script>
      endsubmit ;
      rc = preview('FILE', '_WEBOUT') ;
      rc = preview('CLEAR') ;
      return ;
   end ;
   whcls = 'cmid = ' || quote(sclid) ;
   rc = where(dsid, whcls) ;
   rc = attrn(dsid, 'any') ;  /* If observations match where, rc=1 */
   if (rc ne 1) then do ;
      submit ;
      <script>
         alert("WARNING! No Records Found\nEnter a New CMID") ;
         document.idform.htmlid.value = '' ;
         document.idform.htmlid.focus() ;
      </script>
      endsubmit ;
      rc = preview('FILE', '_WEBOUT') ;
      rc = preview('CLEAR') ;
      rc = where(dsid, 'clear') ;
      return ;
   end ;
   rc = fetch(dsid) ;
   if (rc = 0) then do ;   /* Display results in phone number boxes */
      submit ;
         /* SAS ODS procedure output will be redirected to the user's */
         /* browser; so, there is no need to redirect the contents of the */
         /* preview buffer to _webout at the end of this submit block */
         ods html body = _webout
                  style = statdoc ;
         proc print data = npc.phonebrs label noobs ;
            var tela teln atela ateln ;
            where cmid = "&sclid" ;
         run ;
         ods html close ;
      endsubmit ;
   end ;
return ;
```

→

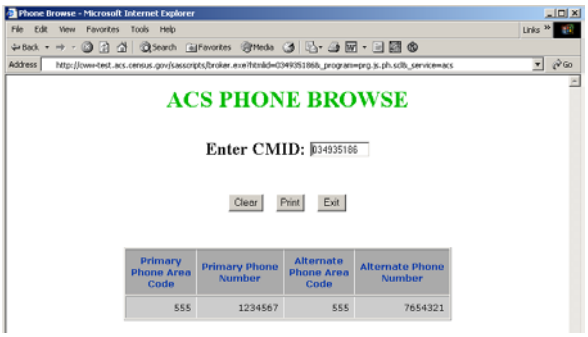

Figure 11. Web Version of Displayed Results – Alternative

CONCLUSION

This paper has presented an uncomplicated technique for reworking a SAS/AF program into an application that functions via the Internet. In the face of such modernization, however, it is still possible for developers to reuse much of their existing SCL code, which provides for a rather simple conversion. By strategically integrating HTML, JavaScript, and SAS programming logic within an SCL program, it is possible to fashion a web-based application that replicates the same look and feel of the corresponding FRAME entry, as well as closely models the usability and functionality of its SAS/AF counterpart.

Specifically, HTML should be used for creating the basic interface that is rendered though the client's web browser. JavaScript is a valuable component necessary for building in enhanced functionality, such as the validation of user input for client-side processing and the generation of warning and error alert messages. While JavaScript can also be programmed to populate (perhaps disabled) HTML input fields for displaying program results, the SAS ODS is an auspicious alternative for developers who are dealing with considerable amounts of output data or for those who are simply trying to generate procedure output that is more colorful and rich in content. Despite the replacement of the FRAME entry with a matching web interface, the underlying SCL code that is responsible for processing user input and handling data set manipulation will remain more or less the same in the updated version.

Certainly, there do exist alternative approaches to this type of web migration, some of which may even be more favorable under different circumstances; nonetheless, by applying the methodology of the discussed approach, programmers can convert their SAS/AF applications to function through the Internet in a seamless and straightforward manner while still being able to preserve much of the underlying SCL code that make up these programs. In doing so, organizations can maintain legacy applications and remain viable in the ever-changing business and technological world of the Information Age.

RESOURCES

SAS Institute Inc. "Base SAS®," *Base SAS® Community*. 2003. <http://support.sas.com/rnd/base/>

SAS Institute Inc. *SAS® Component Language: Reference, Version 8,* Cary, NC: SAS Institute Inc., 1999. 812 pp.

SAS Institute Inc. *SAS® Screen Control Language: Reference, Version 6, Second Edition*, Cary, NC: SAS Institute Inc., 1994. 648 pp.

SAS Institute Inc. "SAS® Stored Processes," *SAS® 9.1 Integration Technologies Developer's Guide*. 2003. <http://support.sas.com/rnd/itech/doc9/dev_guide/stprocess/sessions.html>

SAS Institute Inc. *SAS® Web Tools: SAS/IntrNet® Administration, SAS Instructor-Based Training Course Notes*, *Book Code 57697*, Cary, NC: SAS Institute Inc., 2001. 216 pp.

Turner, Jonah P. "A Pinch of SAS®, a Fraction of HTML, and a Touch of JavaScript Serve Up a Grand Recipe," *Proceedings of the Twenty-Eighth Annual SAS® Users Group International Conference*, Paper 34-28, 2003.

ACKNOWLEDGEMENTS

The author would sincerely like to thank John Stiller for his invaluable guidance and encouragement throughout this entire learning process, Kenneth Dawson for his responsive network support and technical assistance, and, of course, Barbara Diskin for endorsing a work environment that fosters creativity and originality, without which would not have made this whole project possible.

CONTACT INFORMATION

Jonah P. Turner
United States Census Bureau
4700 Silver Hill Road, Mail Stop 7500
Washington, DC 20233-7500
(301) 763-5420 or jonah.p.turner@census.gov