Paper 009-2007

# Spicing Up SAS/IntrNet® Applications (with Additives)

Jonah P. Turner, International Trade Administration, Washington, D.C.

## ABSTRACT

With the advent of SAS/IntrNet software, programmers can extend their SAS applications online to cross-platform, global end-users. The paper "Spicing Up SAS/IntrNet® Applications," published in the proceedings for SUGI 30, demonstrated a variety of techniques for improving such systems by way of enriching procedure output and enhancing interface functionality, as well as simplifying code generation and streamlining program design. In particular, methods were shown for adding vertical scrollbars to resulting tables, conditionally formatting output data, and embedding dynamic links that point to additional information or even call upon other programs. Developers were also presented with ways to integrate both the front- and back-end processes of an Internet application into a single SAS program, thereby fostering a system that is easier to manage and remains up to date.

While all these concepts are still covered in this updated edition, other interesting techniques for spicing up SAS Web-based systems have been included. These additives will help programmers design more sophisticated and user-friendly SAS/IntrNet applications that continue to support a high level of manageability and performance. Specifically, developers can learn to incorporate enhanced navigation into their Web interfaces, in addition to pre-loading images and text that alert users of long-running processes. Certain routines may also be added to produce printer-friendly reports, such that tables and scrollable regions are printed in full without headers, footers, or other select content in display. Ultimately, the extensive collection of techniques illustrated in this latest version provide for an application design that caters to the usability and efficiency needs of all end-users and programmers alike.

## INTRODUCTION

The majority of SAS/IntrNet applications are made up of two components: an HTML graphical user interface (GUI) and a back-end SAS program. The front-end GUI usually consists of text fields, radio buttons, selection boxes, and other form elements that are designed to capture user input. JavaScript may also be embedded into an HTML document to add interactivity to the page and validate information that a user enters, which in effect helps to reduce server-side processing. After an HTML form has been properly completed and submitted, the subsequent Internet request is handled by the Application Dispatcher. Both pieces of the Dispatcher, namely, the Application Broker and the Application Server, work in part to invoke a particular SAS program. The front-end form field entries, which are then defined as global macro variables to the background SAS program, can be used to control program execution, as well as manipulate the generated output that is ultimately returned and rendered in the user's browser.

Although the development of SAS/IntrNet applications is, for the most part, straightforward, it can be rather difficult to design programs that are more functional for end-users. The various types and styles of reports that are generated with these Web-based applications must be readable and comprehensible in order for them to be effective. Having to scroll through many pages of data can pose a burden on those users who need to focus their attention on particular groups of records. In some circumstances, these Web reports are too crowded with information, making it hard for users to point out noteworthy content. For such cases, displaying only vital data in scrollable regions with links to detailed sections would better suffice. Users also often have trouble identifying certain records that may be of greater interest. By conditionally formatting these observations with different fonts, sizes, colors, and images, the resulting Web reports become less complicated to read through and understand. Additionally, media-specific styles can be applied to offer users printer-friendly reports in which extraneous Web content is not included in the printout.

Programmers are faced with many challenges in designing SAS/IntrNet applications that are more manageable on their end as well. Maintaining an HTML form to be consistent with the data that is used by the back-end program is a demanding, yet critical task. If the values that populate the HTML selection boxes, text fields, and other form elements do not correspond to those of the background data, users may find themselves running queries that wastefully consume system resources and even produce erroneous results. Moreover, whenever individual changes are made to the front-end HTML interface, the back-end SAS program, or the underlying data, it is often the case that every one of these components has to be adjusted accordingly. Thus, by dynamically building both the GUI and procedure output commands within a single SAS program, it is possible to create an application that is easier to manage and always current. This approach can in fact be followed to streamline both new and existing applications.

The content discussed herein will focus on several different strategies for resolving the various usability and maintenance issues common among SAS applications that function online. It should be noted that the described techniques are not required for developing workable SAS/IntrNet systems; however, with such enhancements, both end-users and programmers will find these Web-based applications to be more efficient and effective in operation.

## BACKGROUND

To get a better understanding of how SAS, HTML, and JavaScript are used together in the development of SAS/IntrNet applications, it is recommended that the reader review the other pieces in this series. The paper "A Pinch of SAS®, a Fraction of HTML, and a Touch of JavaScript Serve Up a Grand Recipe" published in the SUGI 28 proceedings, provides an introductory approach to creating programs that function online. Its follow-up, "A Recipe for Success: Migrating SAS/AF® Applications to the Web Using SAS®, HTML, and JavaScript," issued for SUGI 29, expounds on the subject by suggesting a workable method for modernizing SAS windowing programs while still maintaining legacy code. Presented as a tutorial for SUGI 31, "Dishing Up SAS/IntrNet® Applications: A Second Helping of a Grand Recipe" offers a step-by-step guide for beginning programmers. These papers all show similar techniques for easily building interactive applications that bring to bear the power of SAS to Internet users.

## IMPLEMENTATION

- ***Problem***

The example to be discussed involves a made-up SAS/IntrNet application referred to as the "Employee Annual Timesheet Electronic Reporting System" or "EATERS" for short. The front-end GUI consists of a form with a selection box containing a list of the names of all current Company X employees. The user, who is typically a manager or a member of the Human Resources Department (HRD), can choose a name from the drop-down list to generate a given employee's yearly time schedule. This resulting Web report includes a breakdown of the time the selected employee has spent in the office, away at lunch, and on leave (i.e. sick, personal, vacation) for each day of the year, along with any relevant notes. Despite the usefulness of this application in operation, there are several problems with the current setup. On the whole, the generated Web timesheet reports are simply too congested with data, making it difficult for users to pinpoint and absorb noteworthy content, such as those observations where an employee has recorded working overtime. It has also been a painstaking task for the developers to continually keep the hard-coded HTML selection box values up to date, considering the number of employees that coincides with Company X's high turnover rate, in addition to managing the front- and back-end portions of this programming assignment. These issues and other related problems are more closely discussed below within the detailed sections.

The HTML code used to fashion the front-end GUI and the original background SAS program designed to generate the employee annual time schedule reports are illustrated here with corresponding screenshots of resultant output:

***Employee Annual Timesheet Electronic Reporting System: Front-end (eatgui.html)***

```
<HTML>
<HEAD>
  <TITLE>EATERS</TITLE>
  <SCRIPT>
    function chkForm() {
      if (document.employees.employee.selectedIndex == 0) {
        alert("Please make a selection.") ;
        return false ;
      } else return true ;
    }
  </SCRIPT>
</HEAD>
<BODY>
  <CENTER>
  <H1>Employee Annual Timesheet Electronic Reporting System</H1>
  <HR><BR><BR><BR>
  <FORM NAME=employees METHOD=get ACTION=
  "/sasscripts/broker.exe" onSubmit="return chkForm();">
    <SELECT NAME=employee>
      <OPTION VALUE=none SELECTED>(Please make a selection)
      <OPTION VALUE="John Adams">Adams, John
      <OPTION VALUE="Chester Arthur">Arthur, Chester
      …
      <OPTION VALUE="George Washington">Washington, George
      <OPTION VALUE="Woodrow Wilson">Wilson, Woodrow
    </SELECT>
    <INPUT TYPE=submit VALUE=Submit>
    <INPUT TYPE=hidden NAME=_program VALUE=hrprg.eatall.sas>
    <INPUT TYPE=hidden NAME=_service VALUE=default>
    <INPUT TYPE=hidden NAME=_debug VALUE=0>
  </FORM>
  </CENTER>
</BODY>
</HTML>
```

→

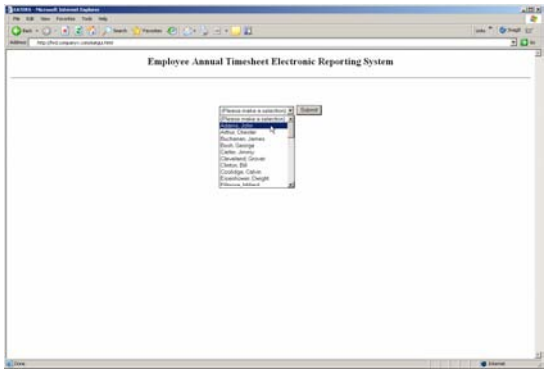
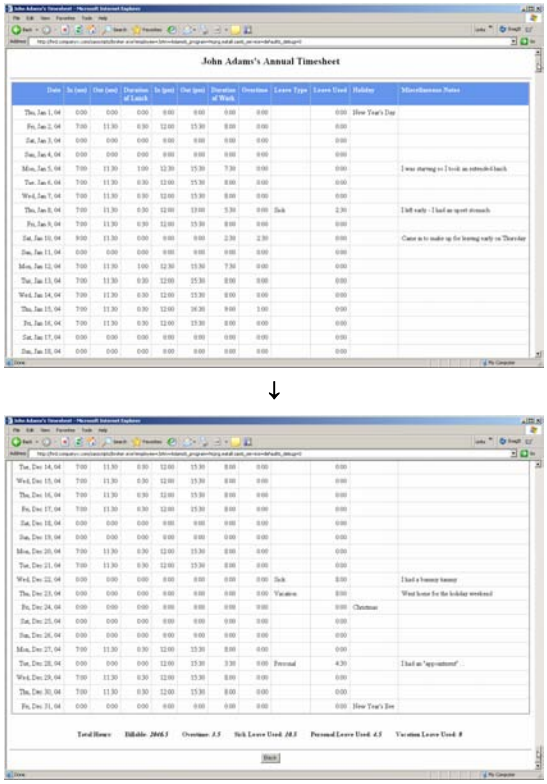



Figure 1. Original interface using HTML and JavaScript

***Employee Annual Timesheet Electronic Reporting System: Back-end (eatall.sas)***

```
%macro eatall ;

  /* Output the title */
  data _null_ ;
    file _webout ;
    put '<HTML>' ;
    put '<HEAD>' ;
    put "<TITLE>&employee's Timesheet</TITLE>" ;
    put '</HEAD>' ;
    put '<BODY>' ;
    put "<H2 ALIGN=center>&employee's Annual Timesheet</H2>" ;
    put '<HR>' ;
  run ;

  /* Output the annual timesheet */
  ods html body=_webout style=sasweb ;
  proc print data=hrdat.timesheets noobs label ;
    where employee="&employee" ;
    var date inam outam lunch inpm outpm work
        overtime leavetype leaveused holiday notes ;
    label date='Date'
          inam='In (am)'
          outam='Out (am)'
          lunch='Duration of Lunch'
          inpm='In (pm)'
          outpm='Out (pm)'
          work='Duration of Work'
          overtime='Overtime'
          leavetype='Leave Type'
          leaveused='Leave Used'
          holiday='Holiday'
          notes='Miscellaneous Notes' ;
    format date weekdate15. inam outam lunch inpm
           outpm work overtime leaveused hhmm. ;
  run ;
  ods html close ;

  /* Calculate the total work and overtime hours */
  proc means data=hrdat.timesheets ;
    where employee="&employee" ;
    var work overtime ;
    output out=totals1 sum=totwork totovertime ;
  run ;
  data _null_ ;
    set totals1 ;
    call symput('totwork', totwork / 3600 ) ;
    call symput('totovertime', totovertime / 3600 ) ;
  run ;

  /* Calculate the total sick, personal, and vacation leave hours used */
  proc means data=hrdat.timesheets ;
    where employee="&employee" ;
    class leavetype ;
    var leaveused ;
    output out=totals2 sum=leaveused ;
  run ;
  data _null_ ;
    set totals2 ;
    if leavetype='Sick' then call symput('totslu', leaveused / 3600 ) ;
    else if leavetype='Personal' then call symput('totplu', leaveused / 3600 ) ;
    else if leavetype='Vacation' then call symput('totvlu', leaveused / 3600 ) ;
  run ;

  /* Output the totals and a back button */
  data _null_ ;
    file _webout ;
    put '<CENTER>' ;
    put '<TABLE CELLPADDING=10 CELLSPACING=10><TR>' ;
    put '<TD><B>Total Hours:</B></TD>' ;
    put "<TD><B>Billable: <I>&totwork</I></B></TD>" ;
    put "<TD><B>Overtime: <I>&totovertime</I></B></TD>" ;
    put "<TD><B>Sick Leave Used: <I>&totslu</I></B></TD>" ;
    put "<TD><B>Personal Leave Used: <I>&totplu</I></B></TD>" ;
    put "<TD><B>Vacation Leave Used: <I>&totvlu</I></B></TD>" ;
    put '</TR></TABLE>' ;
    put '<HR>' ;
    put '<INPUT TYPE=button VALUE=Back
         onClick="javascript:history.back();">' ;
    put '</CENTER>' ;
    put '</BODY>' ;
    put '</HTML>' ;
  run ;

%mend ;
%eatall ;
```




Figures 2-3. Original output using
the SAS ODS with embedded HTML and JavaScript

- ***Solution***

By exploring the various programming issues separately and building the respective solutions into the application one at a time, it will become clear as to how each technique can be exploited in real-world systems that function online. The final version of this SAS/IntrNet application will work in a similar manner and produce comparable results as the original; however, the enhancements made will provide for a system design that brings together the front- and back-end processes into a single SAS program – one that is easier to maintain and a lot more functional.

### ***Issue 0***

An easy, yet constructive technique for preparing a more flavorful application is to simply spice up its name. Surely, a new title such as the "Employee Annual *Thyme*sheet Electronic Reporting System" would better suffice. ;)

### ***Issue 1***

An apparent downside of the original application is that the time schedule Web reports, which include a record for every day of the year, take up too much space. Users who are primarily concerned with viewing the calculated totals find it cumbersome to have to scroll down to the bottom of the page each time to explore these results. Furthermore, depending on the length of the values assigned to certain variables (e.g. *date*, *holiday*, *notes*), the column widths seem to adjust accordingly; thus, a table may look different in the browser every day it gets updated.

When using the ODS, it can be a challenge to predict and control how procedure output will look when displayed in a Web browser given that SAS automatically generates the HTML code to be rendered. Developers can instead strategically place PUT statements within a DATA step to specify the exact HTML code that is to be written out to the source file. While some HTML code may be written out with each iteration of the DATA step, certain start tags and relevant attributes should only be applied during the first iteration and corresponding end tags on the final pass. For example, to output an entire data set, one would open and close an HTML *table* element on the first and last pass of a DATA step using the <TABLE> and </TABLE> tags, respectively, and affix a new row for each iteration:

```
data _null_ ;
  file file ;
  set dataset end=last ;
  if _n_=1 then put '<TABLE>' ;
  put '<TR><TD>' variable1 '</TD><TD>' variable2 '</TD><TD>' … '</TD><TD>' variableN '</TD></TR>' ;
  if last then put '</TABLE>' ;
run ;
```

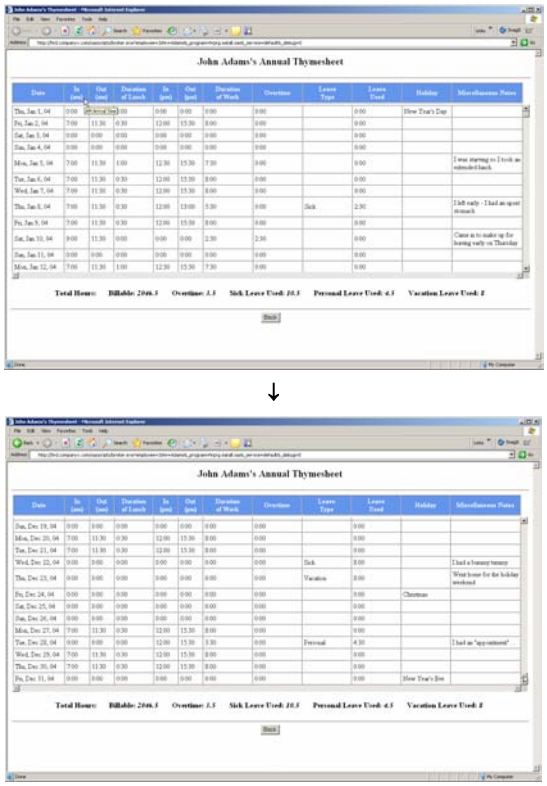
With some HTML knowledge and a little bit of creativity, developers can take advantage of this technique to completely control the output of their SAS/IntrNet programs. In this case, placing a time schedule report in a scrollable region would allow for the summary information listed below the table to be instantly observable. In conjunction with certain Cascading Style Sheet (CSS) properties, like *width*, *height*, and *overflow*, the <DIV> tag can be applied to define a scrollable region. To achieve precise column widths and specify other unique attributes to a column or group of columns, the <COLGROUP> and <COL> tags should be used as part of a *table* element. Here is a way for building an HTML table in a scrollable region that can be used in lieu of procedure output commands:

```
data _null_ ;
  file file ;
  set dataset end=last ;
  if _n_=1 then do ;
    put '<TABLE><TR><TD>' ;  /* This outer table ensures that both inner tables adjust to the browser window and are aligned */
    put '<TABLE>' ;  /* This inner table holds the column headers */
    put '<COLGROUP>' ;
    put '<COL SPAN=number_of_columns_to_span WIDTH=column_header_width>' ;
    /** Add another <COL> for each unique column grouping **/
    put '</COLGROUP>' ;
    put '<TR>' ;
    put '<TD>column_header_for_variable1</TD>' ;
    put '<TD>column_header_for_variable2</TD>' ;
    …
    put '<TD>column_header_for_variableN</TD>' ;
    put '</TR>' ;
    put '</TABLE>' ;
    put '</TD></TR><TR><TD>' ;
    put '<DIV STYLE="width:table_width; height:table_height; overflow:scroll;"><TABLE>' ;  /* This inner table holds the records */
    put '<COLGROUP>' ;
    put '<COL SPAN=number_of_columns_to_span WIDTH=column_width>' ;
    /** Add another <COL> for each unique column grouping **/
    put '</COLGROUP>' ;
  end ;
  put '<TR><TD>' variable1 '</TD><TD>' variable2 '</TD><TD>' … '</TD><TD>' variableN '</TD></TR>' ;
  if last then do ;
    put '</TABLE></DIV>' ;
    put '</TD></TR></TABLE>' ;
  end ;
run ;
```

For the updated version of the EATERS, the PROC PRINT has been replaced with some DATA step processing to set the selected employee's timesheet in a scrollable region with fixed dimensions. The user can now view all the summary information once the page loads without having to scroll to the bottom of the Web page. Longer records will also wrap inside respective table columns instead of forcing them to expand. CSS properties can be applied globally to a document via the *style* element, which is how the developers matched the blue and white color scheme of the table headers in the timesheet. Other special attributes can be included to increase usability of the page, like TITLE= which presents text mouseovers for several HTML elements. With full control of the code produced in the DATA step, one can craft a Web report that closely resembles and is more enhanced than what the ODS can offer:

***Employee Annual Thymesheet Electronic Reporting System: Back-end (eatall.sas)***

```
%macro eatall ;
  /* Output the annual timesheet in a scrollable region */
  data _null_ ;
    file _webout ;
    set hrdat.timesheets end=last ;
    where employee="&employee" ;
    if _n_=1 then do ;
      put '<HTML>' ;
      put '<HEAD>' ;
      put "<TITLE>&employee's Thymesheet</TITLE>" ;
      put '<STYLE>' ;
      put '#header { background-color:#6495ED; color:white; }' ;
      put '</STYLE>' ;
      put '</HEAD>' ;
      put '<BODY>' ;
      put "<H2 ALIGN=center>&employee's Annual Thymesheet</H2>" ;
      put '<HR>' ;
      put '<TABLE ALIGN=center><TR><TD>' ;
      put '<TABLE CELLPADDING=4 BORDER=1>' ;
      put '<COLGROUP>' ;
      put '<COL SPAN=1 WIDTH=75>' ;
      put '<COL SPAN=2 WIDTH=35>' ;
      put '<COL SPAN=1 WIDTH=70>' ;
      put '<COL SPAN=2 WIDTH=35>' ;
      put '<COL SPAN=5 WIDTH=70>' ;
      put '<COL SPAN=1 WIDTH=150>' ;
      put '</COLGROUP>' ;
      put '<TR ID=header ALIGN=center>' ;
      put '<TD TITLE="Date">Date</TD>' ;
      put '<TD TITLE="AM Arrival Time">In<BR>(am)</TD>' ;
      put '<TD TITLE="AM Departure Time">Out<BR>(am)</TD>' ;
      put '<TD TITLE="Total Lunch">Duration<BR>of Lunch</TD>' ;
      put '<TD TITLE="PM Arrival Time">In<BR>(pm)</TD>' ;
      put '<TD TITLE="PM Departure Time">Out<BR>(pm)</TD>' ;
      put '<TD TITLE="Total Work">Duration<BR>of Work</TD>' ;
      put '<TD TITLE="Total Overtime">Overtime</TD>' ;
      put '<TD TITLE="Leave Type Used">Leave<BR>Type</TD>' ;
      put '<TD TITLE="Total Leave Used">Leave<BR>Used</TD>' ;
      put '<TD TITLE="National Holiday">Holiday</TD>' ;
      put '<TD TITLE="Extra Message">Miscellaneous Notes</TD>' ;
      put '</TR>' ;
      put '</TABLE>' ;
      put '</TD></TR><TR><TD>' ;
      put '<DIV STYLE="width:100%; height:320; overflow:scroll;">' ;
      put '<TABLE CELLPADDING=4 BORDER=1>' ;
      put '<COLGROUP>' ;
      put '<COL SPAN=1 WIDTH=75>' ;
      put '<COL SPAN=2 WIDTH=35>' ;
      put '<COL SPAN=1 WIDTH=70>' ;
      put '<COL SPAN=2 WIDTH=35>' ;
      put '<COL SPAN=5 WIDTH=70>' ;
      put '<COL SPAN=1 WIDTH=150>' ;
      put '</COLGROUP>' ;
    end ;
    put '<TR><TD>' date '</TD><TD>' inam '</TD><TD>' outam
        '</TD><TD>' lunch '</TD><TD>' inpm '</TD><TD>'
        outpm '</TD><TD>' work '</TD><TD>' overtime
        '</TD><TD>' leavetype '</TD><TD>' leaveused
        '</TD><TD>' holiday '</TD><TD>' notes '</TD></TR>' ;
    if last then do ;
      put '</TABLE></DIV>' ;
      put '</TD></TR></TABLE>' ;
    end ;
    format date weekdate15. inam outam lunch inpm
           outpm work overtime leaveused hhmm. ;
  run ;
  /** Calculate and output the totals as before **/
%mend ;
%eatall ;
```




Figures 4-5. Modified output with a scrollable region using the HTML *div*, *colgroup*, and *col* elements

### ⏩ *Additive 1*

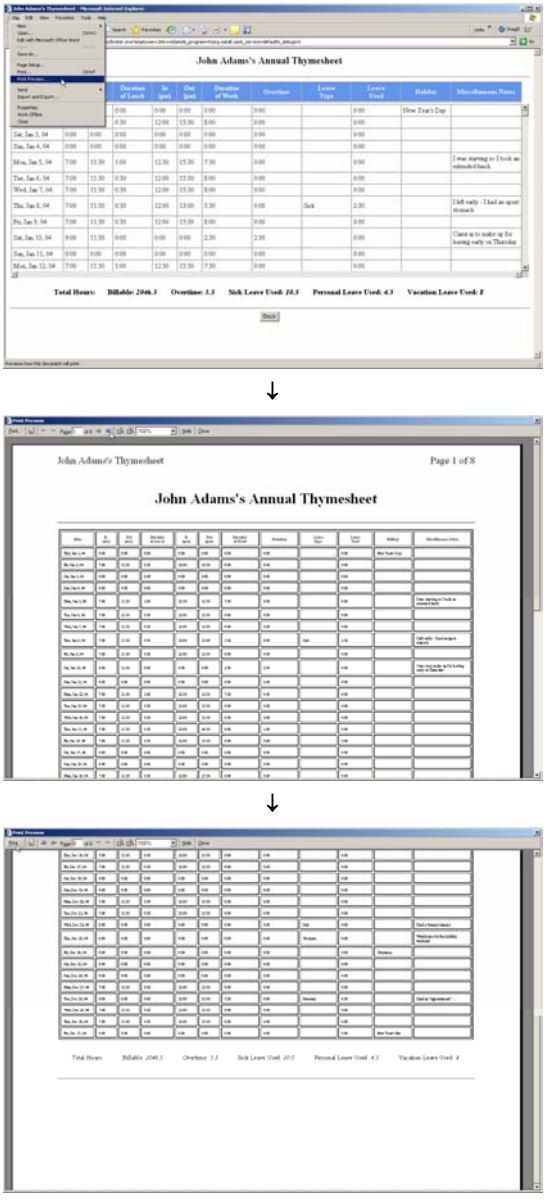
Despite the added convenience of having the tables presented in a scrollable region with fixed dimensions, managers have noticed varying results when attempting to print the annual timesheets from the EATERS. Particularly, they cannot print a time schedule report in its entirety, regardless of the type, version, and settings of their Web browser.

There are some problems associated with the printing of online material, such that content seen on screen may not always fit to paper. The size of text, dimensions of tables, and number of page layers used in a Web page are just a few of the many factors that can have an effect on what is actually printed. As is the case with information stored in a scrollable region, content that is not at once visible on screen will be excluded from a printout by default. These issues, however, can all be resolved by applying unique CSS properties to the printable material only. Using the *@media* rule in conjunction with the "print" media type, programmers can define styles intended only for content to be printed or viewed on screen in print preview mode. For this application, the developers specified two separate styles for the region holding the timesheet: one to display the table in a scrollable region on screen, and the other to render it in full for printing. A media-specific style was also added to exclude the "Back" button from the printout:

***Employee Annual Thymesheet Electronic Reporting System: Back-end (eatall.sas)***

```
%macro eatall ;
  /* Add styles to offer a printer-friendly version of the timesheet */
  data _null_ ;
    file _webout ;
    set hrdat.timesheets end=last ;
    where employee="&employee" ;
    if _n_=1 then do ;
      put '<HTML>' ;
      put '<HEAD>' ;
      put "<TITLE>&employee's Thymesheet</TITLE>" ;
      put '<STYLE>' ;
      put '#header { background-color:#6495ED; color:white; }' ;
      put '#region { width:100%; height:320; overflow:scroll; }' ;
      put '@media print {' ;
      put '#noprint { display:none; }' ;
      put '#region { width:100%; height:100%; overflow:visible; }' ;
      put '}' ;
      put '</STYLE>' ;
      put '</HEAD>' ;
      put '<BODY>' ;
      put "<H2 ALIGN=center>&employee's Annual Thymesheet</H2>" ;
      put '<HR>' ;
      put '<TABLE ALIGN=center><TR><TD>' ;
      put '<TABLE CELLPADDING=4 BORDER=1>' ;
      /** Build the column headers as before **/
      put '</TABLE>' ;
      put '</TD></TR><TR><TD>' ;
      put '<DIV ID=region>' ;
      put '<TABLE CELLPADDING=4 BORDER=1>' ;
      /** Build the colgroup as before **/
    end ;
    put '<TR><TD>' date '</TD><TD>' inam '</TD><TD>' outam
        '</TD><TD>' lunch '</TD><TD>' inpm '</TD><TD>'
        outpm '</TD><TD>' work '</TD><TD>' overtime
        '</TD><TD>' leavetype '</TD><TD>' leaveused
        '</TD><TD>' holiday '</TD><TD>' notes '</TD></TR>' ;
    if last then do ;
      put '</TABLE></DIV>' ;
      put '</TD></TR></TABLE>' ;
    end ;
    format date weekdate15. inam outam lunch inpm
           outpm work overtime leaveused hhmm. ;
  run ;
  /** Calculate the totals as before **/
  /* Output the totals and a back button that is hidden on the printout */
  data _null_ ;
    file _webout ;
    put '<CENTER>' ;
    /** Output the totals as before **/
    put '<HR>' ;
    put '<DIV ID=noprint>' ;
    put '<INPUT TYPE=button VALUE=Back
         onClick="javascript:history.back();">' ;
    put '</DIV >' ;
    put '</CENTER>' ;
    put '</BODY>' ;
    put '</HTML>' ;
  run ;
%mend ;
%eatall ;
```

→




Figures 6-8. Modified printer-friendly output using styles to remove the scrollbars and back button

### *Issue 2*

Even though the EATERS was originally designed to cover all aspects of Company X employees' work schedules, it is not so easy for users to isolate information relevant to their individual needs. Generally speaking, there is too much information being displayed, which can be distracting for managers and HRD personnel. Whereas supervisors are mostly interested in knowing when an employee has taken an extended lunch break or worked less than a full 8-hour day, those from HRD typically use this tool to account for the overtime hours each staff member has earned.

Although it is quite easy to output a data set to the Web, generating a meaningful report can be a difficult task, especially if there is a great deal of information that needs to be displayed. An effective method for highlighting more important fields is to conditionally format output data so that users can easily and quickly discern such content. In other words, programmers can redefine particular records using different HTML tags and attributes that alter the color, size, and style of these values, rather than simply writing out observations in their original state, in order to put greater emphasis on certain output data. This can be done by creating temporary variables that not only maintain the values of the original variables, but also include embedded HTML to control how they should appear when rendered in a Web browser. For instance, a SAS/IntrNet application used to report all Company X employees' salaries would be more insightful if those greater than $100,000 were displayed in green, incomes less than $35,000 were italicized, and salaries equal to the median income of $55,000 were shown in large and bold text for emphasis:

```
data salaries (drop=income rename=(tmpincome=income)) ;
  set hrdat.salaries ;
  if income > 100000 then tmpincome='<FONT COLOR=green>' || income || '</FONT>' ;
  else if income < 35000 then tmpincome='<I>' || income || '</I>' ;
  else if income = 55000 then tmpincome='<BIG><B>' || income || '</B></BIG>' ;
  else tmpincome = income ;
run ;
ods html body=_webout ;
proc print data=salaries ;  run ;
ods html close ;
```

For the EATERS, this technique could certainly be applied to satisfy the specific needs of users. Managers would be able to more quickly identify the days when an employee has taken an extended lunch break or worked less than a full 8-hour day if these records were highlighted in red, bold, and italicized text. Additionally, staff members from HRD would have an easier time sorting through the time schedule reports if records where overtime hours have been documented were displayed with blue, bold, and italicized text. In effect, all users should be able to isolate the vital information they need without much difficulty since these records will now all appear in a more discernible fashion:

***Employee Annual Thymesheet Electronic Reporting System: Back-end (eatall.sas)***

```
%macro eatall ;
  /* Conditionally format records with different text styles */
  data _null_ ;
    file _webout ;
    set hrdat.timesheets end=last ;
    where employee="&employee" ;
    if lunch > 1800 then tmplunch='<FONT COLOR=red>
      <B><I>' || put(lunch, hhmm.) || '</I></B></FONT>' ;
    else tmplunch=put(lunch, hhmm.) ;
    if (mod(date, 7) not in(1, 2) and holiday eq "") then do ;
      if work < 28800 then tmpwork='<FONT COLOR=red>
        <B><I>' || put(work, hhmm.) || '</I></B></FONT>' ;
      else tmpwork=put(work, hhmm.) ;
    end ;
    else tmpwork=put(work, hhmm.) ;
    if overtime > 0 then tmpovertime='<FONT COLOR=blue>
      <B><I>' || put(overtime, hhmm.) || '</I></B></FONT>' ;
    else tmpovertime=put(overtime, hhmm.) ;
    if _n_=1 then do ;
      /** Build the styles, heading, and colgroups as before **/
    end ;
    put '<TR><TD>' date '</TD><TD>' inam '</TD><TD>' outam
        '</TD><TD>' tmplunch '</TD><TD>' inpm '</TD><TD>'
        outpm '</TD><TD>' tmpwork '</TD><TD>' tmpovertime
        '</TD><TD>' leavetype '</TD><TD>' leaveused
        '</TD><TD>' holiday '</TD><TD>' notes '</TD></TR>' ;
    if last then do ;
      put '</TABLE></DIV>' ;
      put '</TD></TR></TABLE>' ;
    end ;
    format date weekdate15. inam outam inpm outpm leaveused hhmm. ;
  run ;
  /** Calculate and output the totals as before **/
%mend ;
%eatall ;
```

→

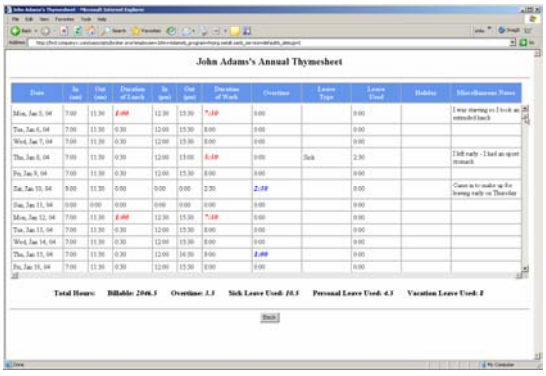



Figure 9. Modified output with conditionally formatted data using temporary variables where extended lunch breaks and short work days are denoted in red, bold, italicized text and recorded overtime hours are denoted in blue, bold, and italicized text

More can be accomplished with this technique of redefining variables to include embedded HTML than simply modifying the color, size, and style of text as it appears in a Web browser. In fact, programmers can mark up output data in any other manner that they are able to do when designing HTML interfaces. For instance, output text can be (conditionally) replaced with meaningful images to reduce space, offer clearer messages, or simply jazz up a Web report. Colorful icons to show the type of leave used help to make the time schedule reports more readable and fun:

***Employee Annual Thymesheet Electronic Reporting System: Back-end (eatall.sas)***

```
%macro eatall ;

  /* Conditionally format leavetype with images */
  data _null_ ;
    file _webout ;
    set hrdat.timesheets end=last ;
    where employee="&employee" ;
    if leavetype='Sick' then tmpleavetype=
      '<IMG SRC=/Images/sick.gif ALT=Sick>' ;
    else if leavetype='Personal' then tmpleavetype=
      '<IMG SRC=/Images/personal.gif ALT=Personal>' ;
    else if leavetype='Vacation' then tmpleavetype=
      '<IMG SRC=/Images/vacation.gif ALT=Vacation>' ;
    /** Conditionally format the other fields as before **/
    if _n_=1 then do ;
      /** Build the styles, heading, and colgroups as before **/
    end ;
    put '<TR><TD>' date '</TD><TD>' inam '</TD><TD>' outam
        '</TD><TD>' tmplunch '</TD><TD>' inpm '</TD><TD>'
        outpm '</TD><TD>' tmpwork '</TD><TD>' tmpovertime
        '</TD><TD>' tmpleavetype '</TD><TD>' leaveused
        '</TD><TD>' holiday '</TD><TD>' notes '</TD></TR>' ;
    if last then do ;
      put '</TABLE></DIV>' ;
      put '</TD></TR></TABLE>' ;
    end ;
    format date weekdate15. inam outam inpm outpm leaveused hhmm. ;
  run ;

  /** Calculate and output the totals as before **/

%mend ;
%eatall ;
```

→

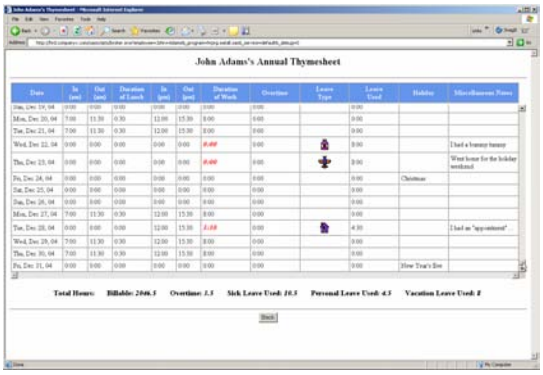



Figure 10. Modified output with conditionally formatted data using temporary variables where "Leave Type" is denoted by these icons:
= sick = personal = vacation

It is apparent that recreating variables with embedded HTML is an effective way to control how output data is to be displayed on the Web. An alternative technique that programmers can use to determine how data set records should be written out to an Internet source file is to build a SAS format consisting of HTML. The FORMAT procedure can instead be applied to replace certain values in the "Leave Type" column with the proper syntax for an HTML image:

***Employee Annual Thymesheet Electronic Reporting System: Back-end (eatall.sas)***

```
%macro eatall ;

  /* $leave is used to conditionally format leavetype with images */
  proc format ;
    value $leave
     'Sick'='<IMG SRC=/Images/sick.gif ALT=Sick>'
     'Personal'='<IMG SRC=/Images/personal.gif ALT=Personal>'
     'Vacation'='<IMG SRC=/Images/vacation.gif ALT=Vacation>' ;
  run ;

  /* Conditionally format leavetype with images */
  data _null_ ;
    file   webout ;
    set hrdat.timesheets end=last ;
    where employee="&employee" ;
    /** Conditionally format the other fields as before **/
    if _n_=1 then do ;
      /** Build the styles, heading, and colgroups as before **/
    end ;
    put '<TR><TD>' date '</TD><TD>' inam '</TD><TD>' outam
        '</TD><TD>' tmplunch '</TD><TD>' inpm '</TD><TD>'
        outpm '</TD><TD>' tmpwork '</TD><TD>' tmpovertime
        '</TD><TD>' leavetype '</TD><TD>' leaveused
        '</TD><TD>' holiday '</TD><TD>' notes '</TD></TR>' ;
    if last then do ;
      put '</TABLE></DIV>' ;
      put '</TD></TR></TABLE>' ;
    end ;
    format date weekdate15. inam outam inpm outpm leaveused hhmm.
           leavetype $leave. ;
  run ;

  /** Calculate and output the totals as before **/

%mend ;
%eatall ;
```

→

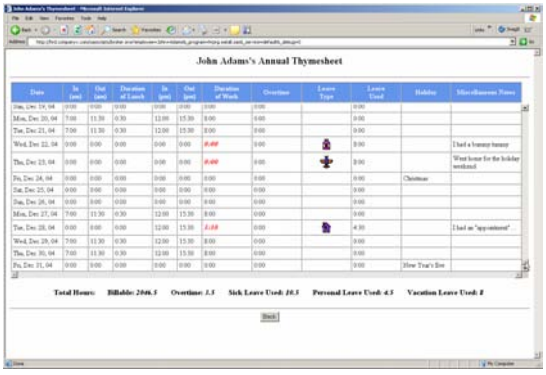



Figure 11. Modified output with conditionally formatted data using PROC FORMAT where "Leave Type" is denoted by these icons:
= sick = personal = vacation

### ⏩ *Additive 2*

Several users have noted that, although it is now easier to isolate information which has been formatted with unique text and images, it is still difficult to focus on entire records of data since every row follows the same color scheme.

People may overlook meaningful records when looking at tabular data containing a lot of repetitive or indistinguishable information. Users of the EATERS cannot always maintain their concentration on individual records in the timesheet because the color scheme of the entire table is the same, with the exception of the header and a few fields that are conditionally formatted with different styles. A simple solution would be to modify the background color of alternating rows in the report so that users do not confuse work hours of one day with those from the next. Using the techniques shown above, CSS properties can be applied conditionally to any combination of cells, columns, or rows in a table. For instance, the programmers could separate weekends, holidays, or the start of each pay period with a new look. In this version, neighboring records are displayed in a slightly different shade:

***Employee Annual Thymesheet Electronic Reporting System: Back-end (eatall.sas)***

```
%macro eatall ;
  /* Conditionally format rows with different background colors */
  data _null_ ;
    file _webout ;
    set hrdat.timesheets end=last ;
    where employee="&employee" ;
    /** Conditionally format all the fields as before **/
    if _n_=1 then do ;
      put '<HTML>' ;
      put '<HEAD>' ;
      put "<TITLE>&employee's Thymesheet</TITLE>" ;
      put '<STYLE>' ;
      put '#header { background-color:#6495ED; color:white; }' ;
      put '#rowA { background-color:#FFFFFF; }' ;
      put '#rowB { background-color:#EEEEEE; }' ;
      put '#region { width:100%; height:320; overflow:scroll; }' ;
      put '@media print {' ;
      put '#noprint { display:none; }' ;
      put '#region { width:100%; height:100%; overflow:visible; }' ;
      put '}' ;
      put '</STYLE>' ;
      put '</HEAD>' ;
      put '<BODY>' ;
      put "<H2 ALIGN=center>&employee's Annual Thymesheet</H2>" ;
      put '<HR>' ;
      put '<TABLE ALIGN=center><TR><TD>' ;
      put '<TABLE CELLPADDING=4 BORDER=1>' ;
      /** Build the colgroup as before **/
      put '<TR ID=header ALIGN=center>' ;
      put '<TD TITLE="Date">Date</TD>' ;
      put '<TD TITLE="AM Arrival Time">In<BR>(am)</TD>' ;
      put '<TD TITLE="AM Departure Time">Out<BR>(am)</TD>' ;
      put '<TD TITLE="Total Lunch">Duration<BR>of Lunch</TD>' ;
      put '<TD TITLE="PM Arrival Time">In<BR>(pm)</TD>' ;
      put '<TD TITLE="PM Departure Time">Out<BR>(pm)</TD>' ;
      put '<TD TITLE="Total Work">Duration<BR>of Work</TD>' ;
      put '<TD TITLE="Total Overtime">Overtime</TD>' ;
      put '<TD TITLE="Leave Type Used">Leave<BR>Type</TD>' ;
      put '<TD TITLE="Total Leave Used">Leave<BR>Used</TD>' ;
      put '<TD TITLE="National Holiday">Holiday</TD>' ;
      put '<TD TITLE="Extra Message">Miscellaneous Notes</TD>' ;
      put '</TR>' ;
      put '</TABLE>' ;
      put '</TD></TR><TR><TD>' ;
      put '<DIV ID=region>' ;
      put '<TABLE CELLPADDING=4 BORDER=1>' ;
      /** Build the colgroup as before **/
    end ;
    if mod(_n_, 2)=1 then put '<TR ID=rowA>' ;
    else put '<TR ID=rowB>' ;
    put '<TD>' date '</TD><TD>' inam '</TD><TD>' outam
        '</TD><TD>' tmplunch '</TD><TD>' inpm '</TD><TD>'
        outpm '</TD><TD>' tmpwork '</TD><TD>' tmpovertime
        '</TD><TD>' tmpleavetype '</TD><TD>' leaveused
        '</TD><TD>' holiday '</TD><TD>' notes '</TD></TR>' ;
    if last then do ;
      put '</TABLE></DIV>' ;
      put '</TD></TR></TABLE>' ;
    end ;
    format date weekdate15. inam outam inpm outpm leaveused hhmm. ;
  run ;
  /** Calculate and output the totals as before **/
%mend ;
%eatall ;
```

→

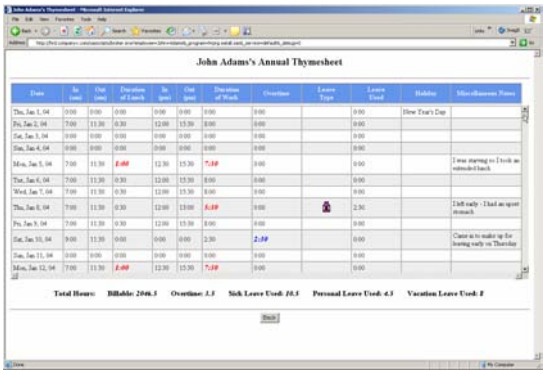



Figure 12. Modified output with conditionally formatted rows using styles to display alternating records with different background colors

### *Issue 3*

The above enhancements have helped to make the time schedule Web reports a little more understandable. Still, some issues with wasted space and overcrowded content exist on account of two rarely used fields. One column, which displays recurring national holidays, is almost always blank, and another field that includes miscellaneous notes left by the employee (e.g. "I left early – I had an upset stomach") is usually empty; however, removing these fields altogether would not satisfy the requirements as most users need to view this information from time to time.

It is common for data that is illustrated in one report to be connected to data displayed in another. When users are able to view all this related content together, they have an easier time analyzing such information. Be that as it may, the more information that gets included in a report, the more crowded and less meaningful it becomes. To resolve this problem in the Web environment, developers can strategically include hyperlinks that point to supplementary information using the HTML *anchor* element. By redefining SAS variables with embedded HTML using the same technique discussed earlier, it is possible to render plain output data as links that point to additional content or even call upon other programs. For example, a SAS/IntrNet application that is used to generate a report listing every Company X employee may be more useful if the text for each person was actually a link to a personal profile page:

```
data staff (drop=person rename=(tmpperson=person)) ;
  set hrdat.staff ;
  tmpperson='<A HREF="/HRD/profiles/' || compress(person) || '.html">' || person || '</A>' ;
run ;
ods html body=_webout ;
proc print data=staff ; run ;
ods html close ;
```

In this case, removing two rarely populated fields would certainly help to reduce the amount of space that is wasted and, in turn, make the Web reports generated by the EATERS more readable. Since the users may still want to view this information on occasion, perhaps to remind themselves of an upcoming holiday or read what "excuse" an employee has marked down for leaving work early, it would be useful to have hyperlinks, one for each day, inserted in the table that point to the supplementary content. Rather than creating another field to contain such links, which would also lead to a loss of valuable space, programmers can instead strategically redefine existing variables with embedded HTML to construct them. For this example, the values already being displayed for the date field can be dynamically transformed into unique links that call upon another SAS/IntrNet program to display the extra information. That program will use the distinct employee and date parameters built into each hyperlink to perform a lookup of the record and return to the user's browser a detailed summary report for the specific day that was chosen. With these hyperlinks in place, the original time schedule reports become less crowded as the fields that were regularly empty are now gone; yet, users can still view this extra content by clicking on the link for a particular day:

***Employee Annual Thymesheet Electronic Reporting System: Back-end (eatall.sas)***

```
%macro eatall ;
  /* Format date with hyperlinks that call a SAS program */
  data _null_ ;
    file _webout ;
    set hrdat.timesheets end=last ;
    where employee="&employee" ;
    tmpdate='<A HREF="broker.exe?employee=' || "&employee" ||
    '&date=' || trim(left(date)) || '&  program=hrprg.eatone.sas
    &_service=default&_debug=0">' || put(date, weekdate15.) || '</A>' ;
    /** Conditionally format the other fields as before **/
    if _n_=1 then do ;
      /** Build the styles and heading as before **/
      put '<FONT COLOR=red><I>* Click on a date to view
           a detailed summary of that record *</I></FONT>' ;
      /** Build the colgroups as before, removing two columns **/
    end ;
    if mod(_n_, 2)=1 then put '<TR ID=rowA>' ;
    else put '<TR ID=rowB>' ;
    put '<TD>' tmpdate '</TD><TD>' inam '</TD><TD>' outam
        '</TD><TD>' tmplunch '</TD><TD>' inpm '</TD><TD>'
        outpm '</TD><TD>' tmpwork '</TD><TD>' tmpovertime
        '</TD><TD>' tmpleavetype '</TD><TD>' leaveused
        '</TD></TR>' ;
    if last then do ;
      put '</TABLE></DIV>' ;
      put '</TD></TR></TABLE>' ;
    end ;
    format inam outam inpm outpm leaveused hhmm. ;
  run ;
  /** Calculate and output the totals as before **/
%mend ;
%eatall ;
```

→

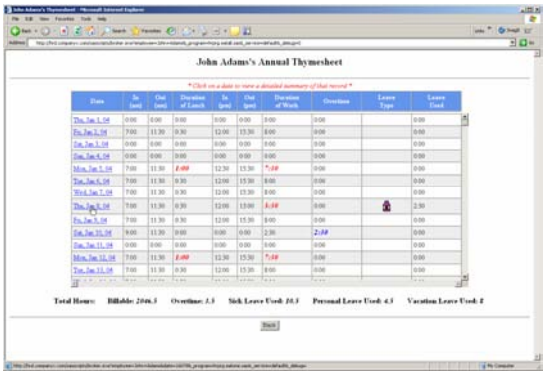



Figure 13. Modified output with formatted data using temporary variables where all date values become hyperlinks that call upon another SAS program to generate a detailed summary report for a selected day

When a user clicks on one of the links, a different SAS program (i.e. *eatone.sas*) gets invoked on the server. The resulting report, a complete time schedule for the particular day chosen, is displayed in the user's Internet browser:

***Employee Annual Thymesheet Electronic Reporting System: Back-end (eatone.sas)***

```
%macro eatone ;
  /* Output the title */
  data _null_ ;
    file _webout ;
    userdate=&date ;
    put '<HTML>' ;
    put '<HEAD>' ;
    put "<TITLE>&employee's Thymesheet</TITLE>" ;
    put '</HEAD>' ;
    put '<BODY>' ;
    put "<H2 ALIGN=center>&employee's Annual Thymesheet</H2>" ;
    put '<H4 ALIGN=center><FONT COLOR=blue>Date:
         </FONT><I>' userdate weekdate15. '</I></H4>' ;
    put '<HR>' ;
  run ;
  /* Output the 1-day detailed summary */
  ods html body=_webout style=sasweb ;
  proc print data=hrdat.timesheets noobs label ;
    where employee="&employee" and date=&date ;
    var inam outam lunch inpm outpm work overtime leavetype leaveused ;
    label inam='In (am)'
          outam='Out (am)'
          lunch='Duration of Lunch'
          inpm='In (pm)'
          outpm='Out (pm)'
          work='Duration of Work'
          overtime='Overtime'
          leavetype='Leave Type'
          leaveused='Leave Used' ;
    format inam outam lunch inpm outpm work overtime leaveused hhmm. ;
  run ;
  ods html close ;
  /* Output the holiday and/or notes (if either field is not empty) */
  data _null_ ;
    file _webout ;
    set hrdat.timesheets ;
    where employee="&employee" and date=&date ;
    if holiday ne ' ' then put '<H4><CENTER><FONT COLOR=blue> *
      National Holiday: </FONT><I>' holiday '</I></CENTER></H4>' ;
    if notes ne ' ' then put '<H4><CENTER><FONT COLOR=blue> *
      Miscellaneous Notes: </FONT><I>' notes '</I></CENTER></H4>' ;
    put '</BODY>' ;
    put '</HTML>' ;
  run ;
%mend ;
%eatone ;
```

→


Figure 14. Detailed output for a particular day using the SAS ODS with embedded HTML and JavaScript

Some managers tend to scrutinize the "excuses" certain employees have for arriving late, taking an extended lunch break, or leaving work early to go home. In order to compare the various notes employees have marked in their time schedules, these managers will open up multiple browsers, one for each day, and arrange them all on their screen. Considering the fact that it takes a lot of processing time to reload a complete time schedule report for each separate browser they need to open, the users have suggested that the detailed records appear in pop-up windows instead. That way, they only have to load the complete annual time schedule once to view multiple single-day observations.

In the same manner that HTML can be sent to an Internet source file using PUT statements in a DATA step, JavaScript can also be written out to better control program output. This scripting language can be used to create dynamic responses, modify the contents of HTML elements, or validate form data before it gets submitted to the server. For security reasons, Company X may require all its employees to log out of the internal Web site because some of the online reports contain sensitive data. JavaScript can then be added to the HTML output of every SAS/IntrNet program to display blinking text in the status bar at the bottom of a user's Web browser as a reminder:

```
data _null_ ;
  file _webout ;
  put '<SCRIPT>' ;
  put 'var blinkrate=250 ;' ;
  put 'var text="Remember to log out before leaving this terminal!!!" ;' ;
  put 'function showText() { window.status=text ;  setTimeout("hideText();", blinkrate) ; }' ;
  put 'function hideText() { window.status="" ;  setTimeout("showText();", blinkrate) ; }' ;
  put 'showText() ;' ;
  put '</SCRIPT>' ;
run ;
ods html body=_webout ;
proc print data=hrdat.passwords ;  run ;
ods html close ;
```

To deal with the managers' need to compare multiple records from a given time schedule all at once, JavaScript can be added to open each single-day report in a different pop-up window. This JavaScript function will be placed with the HTML output using PUT statements in a DATA step. The unique hyperlinks that had been used to directly invoke the program to generate the detailed reports will instead call upon the new JavaScript function when clicked:

***Employee Annual Thymesheet Electronic Reporting System: Back-end (eatall.sas)***

```
%macro eatall ;
  /* Format date with hyperlinks that call a JavaScript function */
  data _null_ ;
    file _webout ;
    set hrdat.timesheets end=last ;
    where employee="&employee" ;
    tmpdate='<A HREF="javascript:popUp(' || "'" || trim(employee) ||
    "'," || trim(left(date)) || ');">' || put(date, weekdate15.) || '</A>' ;
    /** Conditionally format the other fields as before **/
    if _n_=1 then do ;
      /** Build the styles and heading as before **/
      put '<FONT COLOR=red><I>* Click on a date to view
           a detailed summary of that record *</I></FONT>' ;
      /** Build the colgroups as before **/
    end ;
    if mod(_n_, 2)=1 then put '<TR ID=rowA>' ;
    else put '<TR ID=rowB>' ;
    put '<TD>' tmpdate '</TD><TD>' inam '</TD><TD>' outam
        '</TD><TD>' tmplunch '</TD><TD>' inpm '</TD><TD>'
        outpm '</TD><TD>' tmpwork '</TD><TD>' tmpovertime
        '</TD><TD>' tmpleavetype '</TD><TD>' leaveused
        '</TD></TR>' ;
    if last then do ;
      put '</TABLE></DIV>' ;
      put '</TD></TR></TABLE>' ;
      put '<SCRIPT>' ;
      put 'function popUp(person, day) {' ;
      put 'var url="broker.exe?employee=" + person + "&date="
          + day + "&_program=hrprg.eatone.sas&_service=
          default&_debug=0" ;' ;
      put 'popupwin=window.open(url, "", "height=250,
          width=750, scrollbars=yes, resizable=yes") ;' ;
      put '}' ;
      put '</SCRIPT>' ;
    end ;
    format inam outam inpm outpm leaveused hhmm. ;
  run ;
  /** Calculate and output the totals as before **/
%mend ;
%eatall ;
```

→

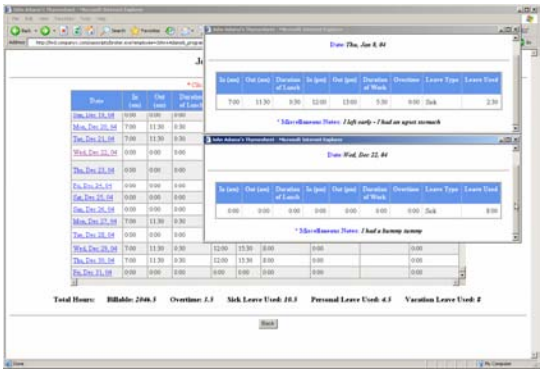


Figure 15. Modified output for multiple days in pop-ups using temporary variables where all date values become hyperlinks that call upon a JavaScript function which executes another SAS program to generate a detailed summary report for a selected day

Any JavaScript function can be added to the output of a SAS/IntrNet application in this fashion. Accordingly, programmers may take advantage of JavaScript *confirm*, *prompt*, and *alert* dialog boxes to report certain messages to users. This is an effective way to save space while still providing users with an ability to view noteworthy content. An alert box reminding Company X staff members of upcoming holidays has been added to the EATERS using this idea. If a holiday is celebrated within the same week a time schedule report has been generated, an alert box informing the user of that information will then be displayed on the screen when the page has finished loading:

***Employee Annual Thymesheet Electronic Reporting System: Back-end (eatall.sas)***

```
%macro eatall ;
  /** Output the annual timesheet as before **/
  /** Calculate and output the totals as before **/
  /* Display an alert box if a holiday is soon approaching */
  data _null_ ;
    file _webout ;
    set hrdat.timesheets ;
    where employee="&employee" and date > today() and
          date <= today() + 7 ;
    if holiday ne "" then do ;
      put '<SCRIPT>' ;
      put 'alert("Reminder: ' holiday ' is only ' _n_ ' days away.
          \n\nPlease remember to mark your calendar!") ;' ;
      put '</SCRIPT>' ;
    end ;
  run ;
%mend ;
%eatall ;
```

→

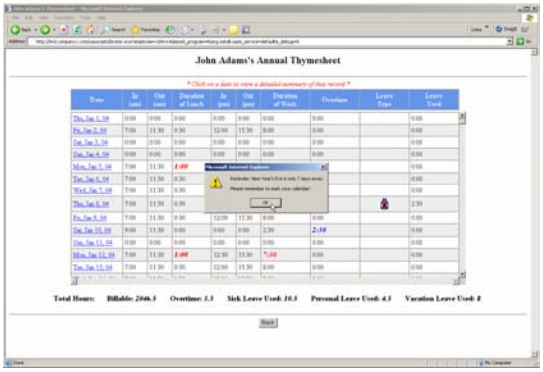



Figure 16. Modified output with a JavaScript alert box using SAS to determine if a holiday is soon approaching

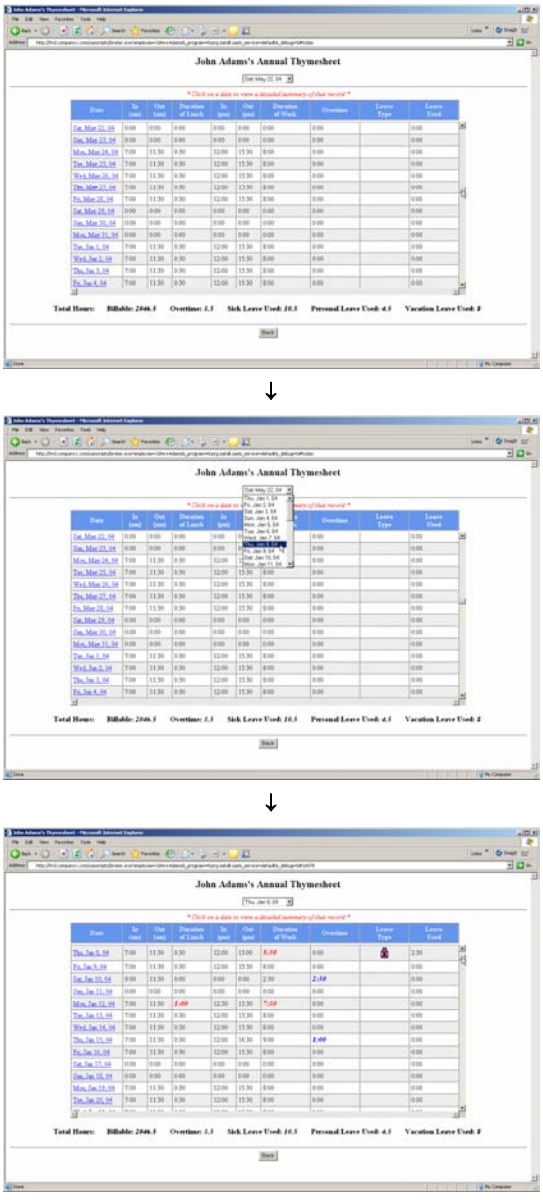
### ⏩ *Additive 3*

The enhancements made to the time schedule reports have greatly improved the usability of the EATERS; however, managers have expressed their frustration in having to carefully scroll through the entire table to find records of importance. In general, the users want to begin looking at an employee's work schedule for the latest pay period. Therefore, having the tables automatically load with the current day as the first viewable record would better suffice.

Developers can further satisfy the needs of many users by adding enhanced navigation into their Web interfaces. For documents containing tabular information, it may be helpful to have the page load directly at a critical data point in the table rather than at the top. To do this, the programmer would need to exploit the abovementioned HTML *anchor* element. Whereas the HREF= attribute refers to the location of a different Web resource, NAME= defines the current anchor so that it may be the destination of another. In order to jump in the page to an anchor identified as <A NAME=*name*>, a link defined as <A HREF=#*name*> would need to be included in the same document. For the EATERS, a unique anchor <A NAME=today> will be output in the same row as the current date, and JavaScript will be invoked whenever the page loads to jump to that record. To improve usability even further, the programmers added a drop-down box at the top of the report for users to quickly navigate to any record in the table. With each iteration of the DATA step, a new anchor is embedded into the row that corresponds to the date of that observation. When the user makes a selection, the JavaScript function goTo() is invoked to relocate the Web page to that record:

***Employee Annual Thymesheet Electronic Reporting System: Back-end (eatall.sas)***

```
%macro eatall ;
  /* Add a selection box for users to jump to any date in the table */
  data _null_ ;
    file   webout ;
    set hrdat.timesheets end=last ;
    where employee="&employee" ;
    if   n  =1 then do ;
      /** Build the styles and heading as before, removing from below **/
      put '<SELECT NAME=date onChange="goTo();">' ;
    end ;
    if date=today() then put '<OPTION VALUE=today SELECTED>'
                             date weekdate15. ;
    else put '<OPTION VALUE=' date '>' date weekdate15. ;
    if last then put '</SELECT>' ;
  run ;
  /* Add anchors and JavaScript to control the table navigation */
  data _null_ ;
    file _webout ;
    set hrdat.timesheets end=last ;
    where employee="&employee" ;
    /** Conditionally format all the fields as before **/
    if _n_=1 then do ;
      /** Build the colgroups as before, removing the styles and heading **/
    end ;
    if mod(  n  , 2)=1 then put '<TR ID=rowA>' ;
    else put '<TR ID=rowB>' ;
    if date=today() then put '<A NAME=today>' ;
    else put '<A NAME=' date '>' ;
    put '<TD>' tmpdate '</TD><TD>' inam '</TD><TD>' outam
        '</TD><TD>' tmplunch '</TD><TD>' inpm '</TD><TD>'
        outpm '</TD><TD>' tmpwork '</TD><TD>' tmpovertime
        '</TD><TD>' tmpleavetype '</TD><TD>' leaveused
        '</TD></TR>' ;
    if last then do ;
      put '</TABLE></DIV>' ;
      put '</TD></TR></TABLE>' ;
      put '<SCRIPT>' ;
      put 'function popUp(person, day) {' ;
      put 'var url="broker.exe?employee=" + person + "&date="
            + day + "&_program=hrprg.eatone.sas&_service=
            default&_debug=0" ;' ;
      put 'popupwin=window.open(url, "", "height=250,
           width=750, scrollbars=yes, resizable=yes") ;' ;
      put '}' ;
      put 'function goTo() {' ;
      put 'window.location="#" + document.all.date.value ;' ;
      put '}' ;
      put 'window.onload=window.location="#today" ;' ;
      put '</SCRIPT>' ;
    end ;
    format inam outam inpm outpm leaveused hhmm. ;
  run ;
  /** Calculate and output the totals as before **/
  /** Display an alert box if a holiday is soon approaching as before **/
%mend ;
%eatall ;
```




Figures 17-19. Modified output with table navigation using embedded HTML and JavaScript to build a new selection box

### *Issue 4*

The HTML front-end of this application contains a drop-down box which holds the name of every Company X employee. Keeping this selection menu up to date, however, is a burdensome task for the developers considering the fact that the list of active employees changes all the time. Consequently, these programmers have to update the HTML form on a regular basis so that staff members can generate time schedule reports for any active employee and also avoid selecting a name from the hard-coded list of someone who is no longer included in the background data.

Just as a report resulting from the execution of a SAS/IntrNet program can be targeted to a user's browser, so can the GUI. Following the abovementioned approach, developers can build a Web interface to replace any front-end *.html* document; essentially, all the HTML code originally used to fashion the GUI should be placed within a SAS program and written out to the Internet source file with PUT statements in a DATA step. Rather than referencing a static *.html* document, users can instead arrive at the GUI by invoking the new program with the appropriate URL. The HTML front-end will then be dynamically generated and rendered in the user's browser. For example, this SAS program can be invoked from a Web browser in place of its *.html* counterpart to get to the company's contact page:

```
data _null_;
  file _webout ;
  put '<HTML>' ;
  put '<HEAD><TITLE>Contact Us</TITLE></HEAD>' ;
  put '<BODY>' ;
  put '<H2 ALIGN=center>Contact Us</H2><HR><BR><BR>' ;
  put '<CENTER><A HREF="mailto:support@companyx.com">Email us a message</A></CENTER><BR><BR><HR><BR>' ;
  put '<CENTER><INPUT TYPE=button VALUE=Back onClick="javascript:history.back();"></CENTER>' ;
  put '</BODY>' ;
  put '</HTML>' ;
run ;
```

Users had previously reached the EATERS front-end by typing the URL *http://hrd.companyx.com/eatgui.html* in their browser's address bar or clicking on a link that referenced this document. Now, users can enter *http://hrd.companyx.com/sasscripts/broker.exe?_program=hrprg.eatgui.sas&_service=default&_debug=0* from the Web address bar or simply click on the original link which has since been modified to point to this specific URL:

***Employee Annual Thymesheet Electronic Reporting System: Front-end (eatgui.sas)***

```
%macro eatgui ;
  /* Output the front-end GUI with a static list of employees */
  data _null_ ;
    file _webout ;
    put '<HTML>' ;
    put '<HEAD>' ;
    put '<TITLE>EATERS</TITLE>' ;
    put '<SCRIPT>' ;
    put 'function chkForm() {' ;
    put 'if (document.employees.employee.selectedIndex == 0) {' ;
    put 'alert("Please make a selection.") ;' ;
    put 'return false ;' ;
    put '} else return true ;' ;
    put '}' ;
    put '</SCRIPT>' ;
    put '</HEAD>' ;
    put '<BODY>' ;
    put '<CENTER>' ;
    put '<H1>Employee Annual Thymesheet Electronic Reporting System</H1>' ;
    put '<HR><BR><BR><BR>' ;
    put '<FORM NAME=employees METHOD=get ACTION=
         "/sasscripts/broker.exe" onSubmit="return chkForm();">' ;
    put '<SELECT NAME=employee>' ;
    put '<OPTION VALUE=none SELECTED>(Please make a selection)' ;
    put '<OPTION VALUE="John Adams">Adams, John' ;
    put '<OPTION VALUE="Chester Arthur">Arthur, Chester' ;
    …
    put '<OPTION VALUE="George Washington">Washington, George' ;
    put '<OPTION VALUE="Woodrow Wilson">Wilson, Woodrow' ;
    put '</SELECT>' ;
    put '<INPUT TYPE=submit VALUE=Submit>' ;
    put '<INPUT TYPE=hidden NAME=_program VALUE=hrprg.eatall.sas>' ;
    put '<INPUT TYPE=hidden NAME=_service VALUE=default>' ;
    put '<INPUT TYPE=hidden NAME=_debug VALUE=0>' ;
    put '</FORM>' ;
    put '</CENTER>' ;
    put '</BODY>' ;
    put '</HTML>' ;
  run ;
%mend ;
%eatgui ;
```

→

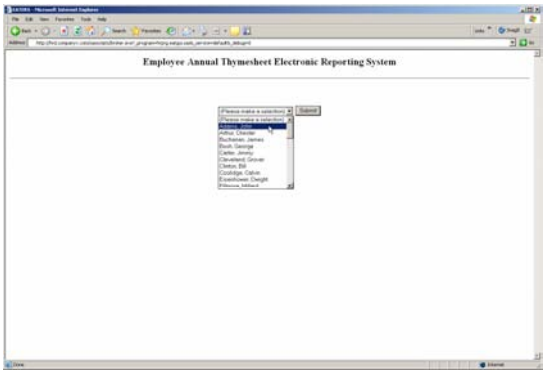



Figure 20. Modified interface with a static list using a SAS program to dynamically generate the HTML and JavaScript that was used to create the original *eatgui.html*

It is easy to dynamically build a front-end GUI by placing code from an *.html* document in a SAS program and having it written out to the Web on each invocation. This technique is especially valuable when certain HTML form fields need to be consistent with the data used in the background. One form field in particular that should parallel the back-end data is an HTML selection box. If new values are ever added or old ones removed from the underlying data, these changes should be made to the front-end selection menu as well. Similar to the approach described earlier for writing out an HTML table to the Web, a selection box can be populated with data set values in a DATA step. In general, a selection box should be opened and closed on the first and last pass of the DATA step, and each option should be written out for each iteration. In the case that the records are not all unique, the data set can be sorted beforehand with the *nodupkey* option so that the duplicate values will not be repeated in the selection menu:

```
proc sort data=dataset out=tmpdataset nodupkey ;  by variable ;  run ;
data _null_ ;
  file file ;
  set tmpdataset end=last ;
  if _n_=1 then put '<SELECT NAME=name>' ;
  put '<OPTION VALUE="' variable '">' variable ;
  if last then put '</SELECT>' ;
run ;
```

To avoid building a timesheet for an individual who is no longer working at Company X and ensure that a report can be generated for each new employee, the selection box on the front-end must contain only the names of all active staff members. Accordingly, the SAS code should be modified so that an option gets added to the drop-down menu for each unique name stored in the background data set. Since a new instance of this program runs each time a user opens the EATERS, the employees listed in the selection box will always mirror whatever is stored in the underlying data. Thus, the programmers do not have to be mindful of employee turnover as names are no longer hard-coded:

***Employee Annual Thymesheet Electronic Reporting System: Front-end (eatgui.sas)***

```
%macro eatgui ;
  /* Sort employees by name in alphabetical order and without duplication */
  proc sort data=hrdat.timesheets out=names nodupkey ;  by lastfirst ;  run ;
  /* Output the front-end GUI with a dynamic list of employees */
  data _null_ ;
    file   webout ;
    set names end=last ;
    if _n_=1 then do ;
      put '<HTML>' ;
      put '<HEAD>' ;
      put '<TITLE>EATERS</TITLE>' ;
      put '<SCRIPT>' ;
      put 'function chkForm() {' ;
      put 'if (document.employees.employee.selectedIndex == 0) {' ;
      put 'alert("Please make a selection.") ;' ;
      put 'return false ;' ;
      put '} else return true ;' ;
      put '}' ;
      put '</SCRIPT>' ;
      put '</HEAD>' ;
      put '<BODY>' ;
      put '<CENTER>' ;
      put '<H1>Employee Annual Thymesheet Electronic Reporting System</H1>' ;
      put '<HR><BR><BR><BR>' ;
      put '<FORM NAME=employees METHOD=get ACTION=
           "/sasscripts/broker.exe" onSubmit="return chkForm();">' ;
      put '<SELECT NAME=employee>' ;
      put '<OPTION VALUE=none SELECTED>(Please make a selection)' ;
    end ;
    put '<OPTION VALUE="' employee '">' lastfirst ;
    if last then do ;
      put '</SELECT>' ;
      put '<INPUT TYPE=submit VALUE=Submit>' ;
      put '<INPUT TYPE=hidden NAME=_program VALUE=hrprg.eatall.sas>' ;
      put '<INPUT TYPE=hidden NAME=_service VALUE=default>' ;
      put '<INPUT TYPE=hidden NAME=_debug VALUE=0>' ;
      put '</FORM>' ;
      put '</CENTER>' ;
      put '</BODY>' ;
      put '</HTML>' ;
    end ;
  run ;
%mend ;
%eatgui ;
```

→

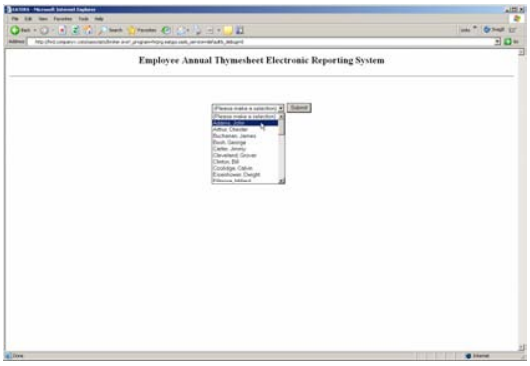



Figure 21. Modified interface with a dynamic list using a SAS program to dynamically generate the HTML and JavaScript that was used to create the original *eatgui.html*

It should be noted that there are alternative methods, such as *htmSQL*, that work to dynamically populate HTML form fields from the front-end; though, these tools require some additional setup and other programming knowledge.

### ⏩ *Additive 4*

The managers are pleased knowing that the drop-down menu listing every Company X employee is always up to date; though, several are unsatisfied with the fact that they have to scroll through hundreds of names to find only those staff members that work in their respective units. It especially becomes a problem when the manager only knows the first or last name of an employee since many people in the organization have similar names. In fact, there exists a unique case of two employees sharing the exact same name, whereby the duplicate is excluded from the list altogether. The programmers realize that breaking down the selection menu by division will resolve all these issues.

Integrating colorful styles, helpful alerts, dynamic links, and enhanced navigation and are just a few of many techniques developers can draw on to improve the overall usability of a Web page. An effective method for designing more user-friendly forms is to break down long selection lists into hierarchical categories so that people have an easier time finding relevant information. Multi-level combo boxes achieve this by allowing users to drill down from general categories to select specific items. With regard to the EATERS, managers would have an easier time finding employees if they could first choose the division for which these individuals work. A drill-down combo control that supports hierarchical selections can be created with HTML and JavaScript. Using arrays, which are practical for storing a set of values in a single variable name, the employees can be grouped together by division. For this application, a multi-dimensional array called *group* has been created, at which each index is stored another array with employees' names for separate divisions. Accordingly, the array stored at *group*[1] holds all employees in Accounting, the array at *group*[2] holds all employees in HR, the array at *group*[3] holds all employees in IT, etc.:

```
group[3]=new Array() ;
group[3][0]=new Option("Adams, John", "John Adams") ;
group[3][1]=new Option("Hoover, Herbert", "Herbert Hoover") ;
group[3][2]=new Option("Jefferson, Thomas", "Thomas Jefferson") ;
group[3][3]=new Option("Kennedy, John", "John Kennedy") ;
group[3][4]=new Option("Lincoln, Abraham", "Abraham Lincoln") ;
group[3][5]=new Option("Madison, James", "James Madison") ;
group[3][6]=new Option("Reagan, Ronald", "Ronald Reagan") ;
group[3][7]=new Option("Roosevelt, Franklin", "Franklin Roosevelt") ;
group[3][8]=new Option("Truman, Harry", "Harry Truman") ;
group[3][9]=new Option("Turner, Jonah", "Jonah Turner") ;
group[3][10]=new Option("Washington, George", "George Washington") ;
```

With JavaScript, a drop-down menu can be dynamically populated with all the options stored inside one of the index arrays based on what a user selects from another. All the necessary code can be produced from a single DATA step:

```
proc sort data=dataset out=tmpdataset nodupkey ; by variable1 variable2 ; run ;
data _null_ ;
  file file ;
  retain v1 0 v2 0 ;
  set tmpdataset end=last ;
  by variable1 ;
  if _n_=1 then do ;
    put '<SCRIPT>' ;
    put 'var group=new Array() ;' ; /* This multi-dimensional array holds an array with all the corresponding options for each group */
    put 'group[0]=new Array() ;' ;
    put 'group[0][0]=new Option("", "none") ;' ;
    put 'function chgForm() {' ; /* This function populates the second menu with all the options that correspond to the group selected from the first one */
    put 'var v1select=document.form.variable1 ;' ;
    put 'var v2select=document.form.variable2 ;' ;
    put 'var v1index=v1select.selectedIndex ;' ;
    put 'for (i=v2select.length; i>=0; i--) v2select.options[i]=null ;' ;
    put 'for (i=0; i<group[v1index].length; i++) v2select.options[i]=new Option(group[v1index][i].text, group[v1index][i].value) ;' ;
    put '}' ;
    put '</SCRIPT>' ;
    put '<FORM NAME=form>' ;
    put '<SELECT NAME=variable1 onChange="chgForm();">' ;
    put '<OPTION VALUE=none SELECTED>(Please make a selection)' ;
  end ;
  if first.variable1 then do ; /* For each group, create an array that contains all the options that correspond to the particular group */
    v1 + 1 ;
    v2 = 0 ;
    put '<OPTION VALUE=' variable1 '>' variable1 ;
    put '<SCRIPT>' ;
    put 'group[' v1 ']=new Array() ;' ;
  end ;
  put 'group[' v1 '][' v2 ']=new Option("' variable2 '", "' variable2 '") ;' ;
  v2 + 1 ;
  if last.variable1 then put '</SCRIPT>' ;
  if last then do ;
    put '</SELECT>' ;
    put '<SELECT NAME=variable2>' ;
    put '<OPTION VALUE=none>' ;
    put '</SELECT>' ;
    /** Add other (hidden) fields as necessary **/
    put '</FORM>' ;
  end ;
run ;
```

Although the above DATA step is moderately complex to interpret, the generated HTML and JavaScript output is much less difficult to comprehend. When configured for the EATERS, the resulting interface consists of two drop-down boxes: one to hold every division in Company X, and another to list all the employees that work for each. When an option is chosen from the first selection menu, the function chgForm() is invoked using the *onChange* JavaScript event handler. This function works by emptying out the list of the second drop-down box and then re-populating it with all the options held in the array corresponding to the division that the user selected. Once the user submits the form, the two items that were chosen get passed as parameters to the background program *eatall.sas*:

***Employee Annual Timesheet Electronic Reporting System: Front-end (eatgui.html)***

```
<HTML>
<HEAD>
  <TITLE>EATERS</TITLE>
  <SCRIPT>
    function chkForm() {
      if (document.employees.division.selectedIndex == 0) {
        alert("Please make a selection.") ;
        return false ;
      } else return true ;
    }
    var group=new Array() ;
    group[0]=new Array() ;
    group[0][0]=new Option("", "none") ;
    function chgForm() {
      var v1select=document.employees.division ;
      var v2select=document.employees.employee ;
      var v1index=v1select.selectedIndex ;
      for (i=v2select.length; i>=0; i--) v2select.options[i]=null ;
      for (i=0; i<group[v1index].length; i++) v2select.options[i]=
      new Option(group[v1index][i].text, group[v1index][i].value) ;
    }
  </SCRIPT>
</HEAD>
<BODY>
  <CENTER>
  <H1>Employee Annual Thymesheet Electronic Reporting System</H1>
  <HR><BR><BR><BR>
  <FORM NAME=employees METHOD=get ACTION=
  "/sasscripts/broker.exe" onSubmit="return chkForm();">
  <SELECT NAME=division onChange="chgForm();">
    <OPTION VALUE=none SELECTED>(Please make a selection)
    <OPTION VALUE="Accounting">Accounting
    <SCRIPT>
      group[1]=new Array() ;
      group[1][0]=new Option("Bush, George", "George Bush") ;
      group[1][1]=new Option("Clinton, Bill", "Bill Clinton") ;
      …
    </SCRIPT>
    <OPTION VALUE="Human Resources">Human Resources
    <SCRIPT>
      group[2]=new Array() ;
      group[2][0]=new Option("Fillmore, Millard", "Millard Fillmore") ;
      group[2][1]=new Option("Ford, Gerald", "Gerald Ford") ;
      …
    </SCRIPT>
    <OPTION VALUE="Information Technology">Information Technology
    <SCRIPT>
      group[3]=new Array() ;
      group[3][0]=new Option("Adams, John", "John Adams") ;
      group[3][1]=new Option("Hoover, Herbert", "Herbert Hoover") ;
      …
    </SCRIPT>
      …
  </SELECT>
  <SELECT NAME=employee>
    <OPTION VALUE=none>
  </SELECT>
  <INPUT TYPE=submit VALUE=Submit>
  <INPUT TYPE=hidden NAME=_program VALUE=hrprg.eatall.sas>
  <INPUT TYPE=hidden NAME=_service VALUE=default>
  <INPUT TYPE=hidden NAME=_debug VALUE=0>
  </FORM>
  </CENTER>
</BODY>
</HTML>
```

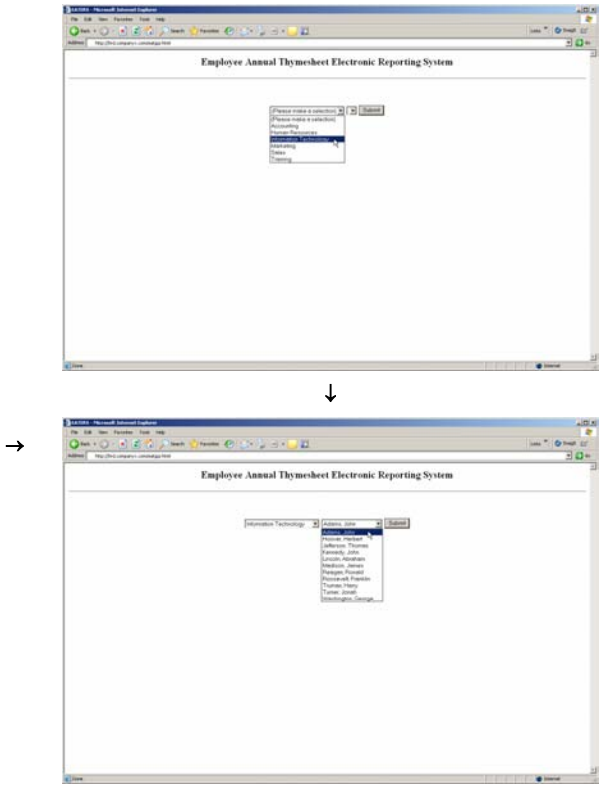



Figure 22-23. Modified interface with combo boxes using JavaScript to store groups of employees by division in arrays and dynamically populate the selection menu

The developers updated the PROC SORT and DATA step in *eatgui.sas* to accommodate the multi-level combo boxes, and the two other programs have been modified to include the division in the query. Now, whenever users choose an item from the *division* drop-down menu, the *employee* selection box is automatically populated with the corresponding options. The duplicate record that was excluded from the output before is readily available because the employees who share the exact same name work in separate divisions. The output is still generated dynamically with each new request, which means the front-end lists will always mirror that which is stored in the background data set. With these changes in place, managers will have an easier time finding only those employees in their unit:

***Employee Annual Thymesheet Electronic Reporting System: Front-end (eatgui.sas)***

```
%macro eatgui ;
  /* Sort employees by division in alphabetical order and without duplication */
  proc sort data=hrdat.timesheets out=names nodupkey ; by division lastfirst ; run ;
  /* Output the front-end GUI with dynamic lists of divisions and employees */
  data _null_ ;
    file _webout ;
    retain v1 0 v2 0 ;
    set names end=last ;
    by division ;
    if _n_=1 then do ;
      put '<HTML>' ;
      put '<HEAD>' ;
      put '<TITLE>EATERS</TITLE>' ;
      put '<SCRIPT>' ;
      put 'function chkForm() {' ;
      put 'if (document.employees.division.selectedIndex == 0) {' ;
      put 'alert("Please make a selection.") ;' ;
      put 'return false ;' ;
      put '} else return true ;' ;
      put '}' ;
      put 'var group=new Array() ;' ;
      put 'group[0]=new Array() ;' ;
      put 'group[0][0]=new Option("", "none") ;' ;
      put 'function chgForm() {' ;
      put 'var v1select=document.employees.division ;' ;
      put 'var v2select=document.employees.employee ;' ;
      put 'var v1index=v1select.selectedIndex ;' ;
      put 'for (i=v2select.length; i>=0; i--) v2select.options[i]=null ;' ;
      put 'for (i=0; i<group[v1index].length; i++) v2select.options[i]=
           new Option(group[v1index][i].text, group[v1index][i].value) ;' ;
      put '}' ;
      put '</SCRIPT>' ;
      put '</HEAD>' ;
      put '<BODY> ' ;
      put '<CENTER>' ;
      put '<H1>Employee Annual Thymesheet Electronic Reporting System</H1>' ;
      put '<HR><BR><BR><BR>' ;
      put '<FORM NAME=employees METHOD=get ACTION=
           "/sasscripts/broker.exe" onSubmit="return chkForm();">' ;
      put '<SELECT NAME=division onChange="chgForm();">' ;
      put '<OPTION VALUE=none SELECTED>(Please make a selection)' ;
    end ;
    if first.division then do ;
      v1 + 1 ;
      v2 = 0 ;
      put '<OPTION VALUE=' division '>' division ;
      put '<SCRIPT>' ;
      put 'group[' v1 ']=new Array() ;' ;
    end ;
    put 'group[' v1 '][' v2 ']=new Option("' lastfirst '", "' employee '") ;' ;
    v2 + 1 ;
    if last.division then put '</SCRIPT>' ;
    if last then do ;
      put '</SELECT>' ;
      put '<SELECT NAME=employee>' ;
      put '<OPTION VALUE=none>' ;
      put '</SELECT>' ;
      put '<INPUT TYPE=submit VALUE=Submit>' ;
      put '<INPUT TYPE=hidden NAME=_program VALUE=hrprg.eatall.sas>' ;
      put '<INPUT TYPE=hidden NAME=_service VALUE=default>' ;
      put '<INPUT TYPE=hidden NAME=_debug VALUE=0>' ;
      put '</FORM>' ;
      put '</CENTER>' ;
      put '</BODY>' ;
      put '</HTML>' ;
    end ;
  run ;
%mend ;
%eatgui ;
```

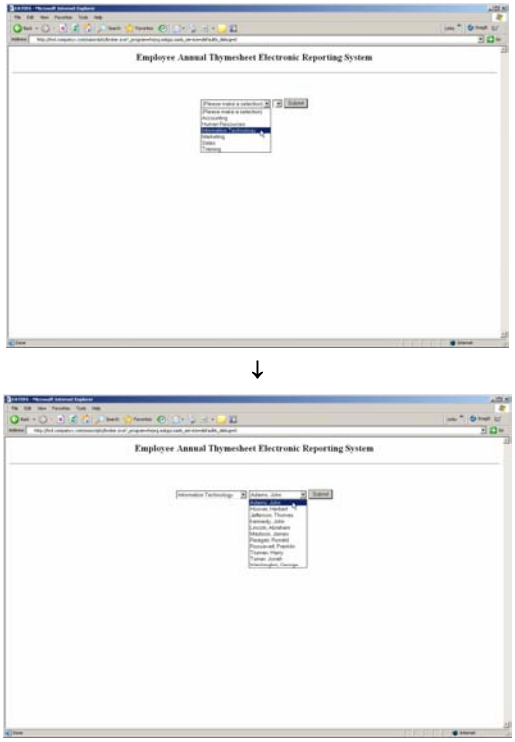



Figure 24-25. Modified interface with dynamic lists using a SAS program to dynamically generate the HTML and JavaScript that was used to create the updated *eatgui.html*

### *Issue 5*

The programmers no longer have to regularly maintain a front-end portion of the EATERS given that the interface is dynamically generated with the underlying data used to populate the drop-down menu of all active employees. Still, these developers find it to be a painstaking task to have to maintain 3 separate SAS programs, namely, *eatgui.sas*, *eatall.sas*, and *eatone.sas*, for just one SAS/IntrNet application. If there are ever any modifications that need to be made to one of these programs, it is usually the case that all of them have to be revised to account for such changes.

It is possible to combine each portion of a SAS/IntrNet application into a single SAS program. Once the front-end *.html* document has been transformed into a SAS program by following the techniques described above, this piece can be conveniently integrated within the same program that is used to build the resulting Web reports. After each function is separated with if-then-else macro logic, a hidden flag variable (e.g. *mode*) can be passed along with every Internet request to control which of the two conditions to execute. The program *proceeds.sas*, shown below, is used to generate both the front-end Web interface and output reports that illustrate the daily profits earned at Company X:

```
/* The URL first called is http://hrd.companyx.com/sasscripts/broker.exe?_program=hrprg.proceeds.sas&_service=default&_debug=0&mode=gui */
%macro proceeds ;

  /* When mode=gui, a front-end interface containing a drop-down menu of each business day is displayed in the user's Web browser */
  %if &mode=gui %then %do ;
    data _null_ ;
      file _webout ;
      put '<HTML>' ;
      put '<HEAD>' ;
      put '<TITLE>Daily Proceeds</TITLE>' ;
      put '</HEAD>' ;
      put '<BODY>' ;
      put '<H2 ALIGN=center>Daily Proceeds</H2><HR>' ;
      put '<CENTER><BR><BR><FORM NAME=sales METHOD=get ACTION="/sasscripts/broker.exe">' ;
      put '<B>Select a day to create a proceeds report:</B>' ;
      put '<SELECT NAME=day>' ;
      put '<OPTION VALUE=Mon>Monday' ;
      put '<OPTION VALUE=Tue>Tuesday' ;
      put '<OPTION VALUE=Wed>Wednesday' ;
      put '<OPTION VALUE=Thu>Thursday' ;
      put '<OPTION VALUE=Fri>Friday' ;
      put '</SELECT>' ;
      put '<INPUT TYPE=submit VALUE=Submit>' ;
      put '<INPUT TYPE=hidden NAME=_program VALUE=hrprg.proceeds.sas>' ;  /* Call this same program when the form is submitted */
      put '<INPUT TYPE=hidden NAME=_service VALUE=default>' ;
      put '<INPUT TYPE=hidden NAME=_debug VALUE=0>' ;
      put '<INPUT TYPE=hidden NAME=mode VALUE=report>' ;  /* Set mode=report when the form is submitted to execute the other condition */
      put '</FORM><BR><HR><INPUT TYPE=button VALUE=Back onClick="javascript:history.back();"></CENTER>' ;
      put '</BODY>' ;
      put '</HTML>' ;
    run ;
  %end ;

  /* When mode=report, the resulting output report of business proceeds for the user-selected day is displayed in the user's Web browser */
  %else %if &mode=report %then %do ;
    %let fn='<INPUT TYPE=button VALUE=Back onClick="javascript:history.back();">' ;
    title "Proceeds report for &day" ;
    ods html body=_webout style=statdoc ;
    proc print data=hrdat.profits_&day noobs ;  footnote &fn ;  run ;
    ods html close ;
  %end ;

%mend ;
%proceeds ;
```

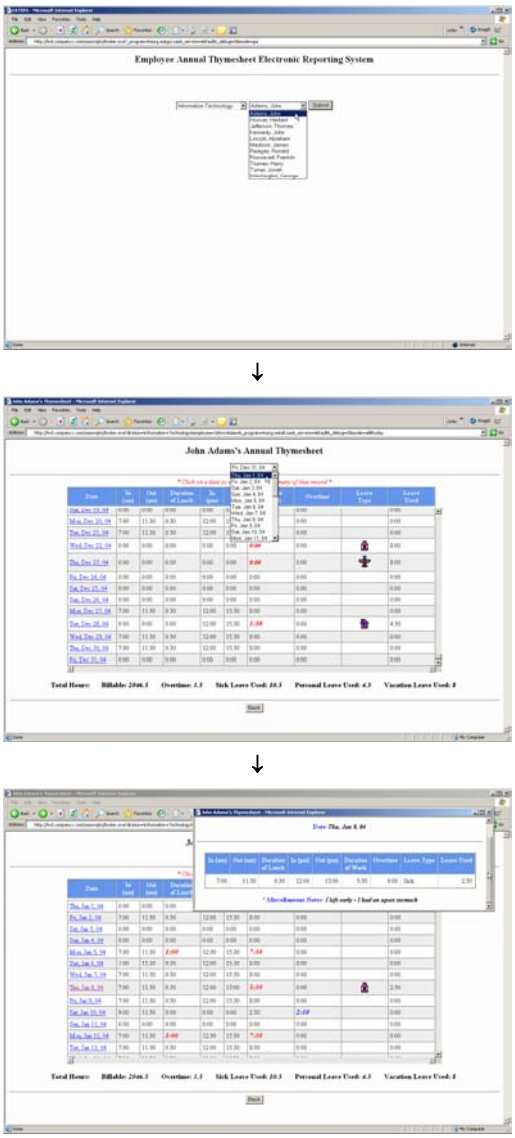
In effect, developers can add any number of unique functions within the same SAS program, yet only execute a particular one (or perhaps more than one) by passing in an appropriately assigned flag variable for each new request:

```
/* Execute a particular function based on the value of the passed flag variable mode */
%macro macro ;

  /* When mode is assigned the value function1, then perform this particular routine */
  %if &mode=function1 %then %do ;
    /** Insert code here for function1 **/
  %end ;

  /* When mode is assigned the value function2, then perform this particular routine */
  %else %if &mode=function2 %then %do ;
    /** Insert code here for function2 **/
  %end ;
  …

  /* When mode is assigned the value functionN, then perform this particular routine */
  %else %if &mode=functionN %then %do ;
    /** Insert code here for functionN **/
  %end ;

%mend ;
%macro ;
```

For this case, the developers would have a much easier time maintaining the EATERS if all 3 programs were merged together (i.e. *eaters.sas*), as opposed to having separate SAS programs to create the GUI (i.e. *eatgui.sas*), build the complete time schedules (i.e. *eatall.sas*), and generate detailed reports for particular days (i.e. *eatone.sas*). The flag variable *mode* should then be passed as a hidden parameter with each new Internet request to control which condition to execute. This parameter can be assigned the value *gui*, *all*, or *one*, depending on whether the user has chosen to arrive at the interface which holds the selection box listing all active employees, produce a time schedule report for a given staff member, or view the detailed summary information for a specific day of work, respectively:

***Employee Annual Thymesheet Electronic Reporting System: Front- and Back-end (eaters.sas)***

```
%macro eaters ;
  /* When mode=gui, output the front-end GUI for selecting an employee */
  /*   program=hrprg.eaters.sas&_service=default&_debug=0&mode=gui */
  %if &mode=gui %then %do ;
    /** Sort the data as before **/
    /* Output the front-end GUI */
    data _null_ ;
      file _webout ;
      retain v1 0 v2 0 ;
      set names end=last ;
      by division ;
      /** Build the heading, first selection box, and JavaScript as before **/
      if last then do ;
        put '</SELECT>' ;
        put '<SELECT NAME=employee>' ;
        put '<OPTION VALUE=none>' ;
        put '</SELECT>' ;
        put '<INPUT TYPE=submit VALUE=Submit>' ;
        put '<INPUT TYPE=hidden NAME=_program VALUE=hrprg.eaters.sas>' ;
        put '<INPUT TYPE=hidden NAME=_service VALUE=default>' ;
        put '<INPUT TYPE=hidden NAME=_debug VALUE=0>' ;
        put '<INPUT TYPE=hidden NAME=mode VALUE=all>' ;
        put '</FORM>' ;
        put '</CENTER>' ;
        put '</BODY>' ;
        put '</HTML>' ;
      end ;
    run ;
  %end ;
  /* When mode=all, output the annual timesheet for the selected employee */
  /*   program=hrprg.eaters.sas&  service=default&_debug=0&mode=all */
  %else %if &mode=all %then %do ;
    /* Output the annual timesheet */
    data _null_ ;
      file _webout ;
      set hrdat.timesheets end=last ;
      where division="&division" and employee="&employee" ;
      /** Build the styles, heading, and scrollable region as before **/
      if last then do ;
        put '</TABLE></DIV>' ;
        put '</TD></TR></TABLE>' ;
        put '<SCRIPT>' ;
        put 'function popUp(person, day) {' ;
        put 'var url="broker.exe?employee=" + person + "&date="
            + day + "&  program=hrprg.eaters.sas&_service=
            default&_debug=0&mode=one" ;' ;
        put 'popupwin=window.open(url, "", "height=250,
            width=750, scrollbars=yes, resizable=yes") ;' ;
        put '}' ;
        put 'function goTo() {' ;
        put 'window.location="#" + document.all.date.value ;' ;
        put '}' ;
        put 'window.onload=window.location="#today" ;' ;
        put '</SCRIPT>' ;
      end ;
      format inam outam inpm outpm leaveused hhmm. ;
    run ;
    /** Calculate and output the totals as before **/
    /** Display an alert box if a holiday is soon approaching as before **/
  %end ;
  /* When mode=one, output a detailed summary record for the selected day */
  /*   program=hrprg.eaters.sas&  service=default&_debug=0&mode=one */
  %else %if &mode=one %then %do ;
    /** Output the detailed summary record as before, using division in the query **/
  %end ;
%mend ;
%eaters ;
```




Figures 26-28. A spiced up SAS/IntrNet application – the joint program *eaters.sas* is used to fashion the GUI, annual timesheets, and detailed daily summaries

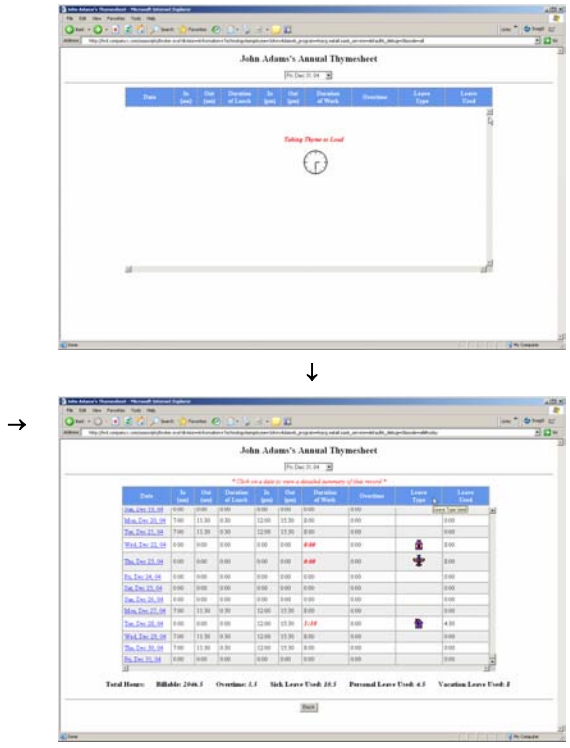
### ⏩ *Additive 5*

The latest version of the EATERS offers a number of features that benefit the users and programmers alike. With other applications being similarly updated, Company X has experienced increased traffic on their network as employees are making better use of these online tools. Now and then, system performance is degraded, which results in certain pages taking more time to load than normal. It is common for employees to assume then that the applications are not working properly. By displaying a pre-loading image and text to the screen whenever a SAS program is running on the server, users will become accustomed to possibly having to wait on their results to appear.

Cascading Style Sheets (CSS) offer a simple way to control the appearance and layout of a Web page in an organized and efficient manner. Developers can take advantage of different CSS properties to set background effects, table dimensions, color schemes, text alignment, and spacing rules, among other things. As demonstrated in previous examples, several of these kinds of styles have been applied to the EATERS to render the timesheets in a scrollable region with fixed dimensions, define text and background colors for the headers and rows, and support the printing of the tables in full. CSS "classification" properties also exist to allow programmers to affect where and how HTML elements should be displayed on the page; specifically, they can be placed in a static, relative, absolute, or fixed position and even hidden from view altogether. Using the *position* and *visibility* CSS properties, the programmers of this application have included both a text message and an animated image to always appear in the middle of the screen to alert users that the time schedule reports may be "Taking Thyme to Load." An animated GIF (i.e. Graphics Interchange Format) is an image file that is composed of many different images, whereby each is shown in a timed sequence to give the effect of a picture in motion. Although special graphic software is needed to create these images, plenty are available to download from the Internet. For the EATERS, an animated clock with moving hands is pictured below the text to signal that the background program is in process. Essentially, whenever the program *eaters.sas* is invoked, the section of HTML code labeled "loading," which is output in the first DATA step, will show up in the user's Web browser immediately. Once the program has finished executing and, subsequently, the page has completed loading, this block is then hidden from view using a JavaScript function call. Users should always expect to see the results on the screen in spite of sometimes having to wait longer than normal:

***Employee Annual Thymesheet Electronic Reporting System: Front- and Back-end (eaters.sas)***

```
%macro eaters ;
  /* Show an image and text while the output is loading */
  data _null_ ;
    file  webout ;
    put '<STYLE>' ;
    put '#loading { position:absolute; left:45%; top:25%; visibility:visible; }' ;
    put '</STYLE>' ;
    put '<DIV ID=loading>' ;
    put '<CENTER>' ;
    put '<FONT COLOR=red><I>Taking Thyme to Load</I></FONT>' ;
    put '<BR><BR>' ;
    put '<IMG SRC=/Images/loading.gif ALT=Loading>' ;
    put '</CENTER>' ;
    put '</DIV>' ;
  run ;

  /* When mode=gui, output the front-end GUI for selecting an employee */
  /* _program=hrprg.eaters.sas&_service=default&_debug=0&mode=gui */
  %if &mode=gui %then %do ;

    /** Output the front-end GUI as before **/

  %end ;

  /* When mode=all, output the annual timesheet for the selected employee */
  /* _program=hrprg.eaters.sas&_service=default&_debug=0&mode=all */
  %else %if &mode=all %then %do ;

    /** Output the annual timesheet as before **/

  %end ;

  /* When mode=one, output a detailed summary record for the selected day */
  /* _program=hrprg.eaters.sas&_service=default&_debug=0&mode=one */
  %else %if &mode=one %then %do ;

    /** Output the detailed summary record as before **/

  %end ;

  /* Hide the image and text after the output has loaded */
  data _null_ ;
    file  webout ;
    put '<SCRIPT>' ;
    put 'window.onload=document.all.loading.style.visibility="hidden" ;' ;
    put '</SCRIPT>' ;
  run ;
%mend ;
%eaters ;
```




Figures 29-30. A truly spiced up SAS/IntrNet application – an image and text now appears whenever the GUI, annual timesheets, and detailed daily summaries are loading

## CONCLUSION

SAS/IntrNet software provides developers with the facility to extend their SAS programs to the Web. With just some basic knowledge of HTML and JavaScript, programmers can create practical applications for producing, updating, and querying SAS data via the Internet. The paper "Spicing Up SAS/IntrNet® Applications," published in the proceedings for SUGI 30, demonstrated a variety of techniques for improving such systems by way of enriching procedure output and enhancing interface functionality, as well as simplifying code generation and streamlining program design. This latest installment contains additives, which can help programmers design more sophisticated and user-friendly SAS/IntrNet applications that continue to support a high level of manageability and performance.

It can be a challenge for programmers to develop Web applications that are functional and appealing. It is common for users to have difficulty evaluating reports that are lengthy and crammed with information. One way of reducing this burden is by adding vertical scrollbars to excessively long tables. In this manner, users may focus their attention on particular groups of records with greater ease and view summary information displayed at the bottom of such tables as soon as they load. Including media-specific styles allow for printer-friendly reports in which scrollable regions are output in full and certain content is excluded from printouts. For the cases where too much information is shown, programmers can conditionally format records with different fonts, sizes, colors, and even graphics, enabling users to more easily and quickly identify noteworthy content. With a similar approach, it is possible for developers to remove and replace supplementary information altogether with links that reference other back-end programs, which may then be called upon to display the extra content as needed. Web interfaces can be enhanced even further with multi-level navigation, and pre-loading images and text that alert users of long-running processes.

SAS developers are faced with many challenges in designing Web-based systems that are more manageable on their end as well. With any Internet application, it is necessary for programmers to reliably update the front-end HTML form fields to match changes that occur in the background data. If these two components are not consistent, users may end up running queries that produce erroneous results and are wasteful of server-side resources. In addition, all the form field options users require to generate certain reports may not be available to them when these items do not reflect the data that has been changed. By replacing the hard-coded front-end *.html* document with a SAS program to dynamically generate the GUI for each request, such problems are eliminated as the form fields will always be populated using the underlying data. Thus, programmers no longer have to maintain the front-end portion of their SAS/IntrNet applications because the GUI will at all times be up to date. Using macro if-then-else conditional logic, developers can merge the front- and back-end processes of an Internet application into a single SAS program to foster a system that is easier to manage and always current. As users will still be able to transparently toggle between different functions, programmers will only have to maintain these separate routines in one joint program.

By applying the various techniques discussed, programmers can rework any SAS/IntrNet application to be more effective in operation. With such enhancements in place, users will have less difficulty examining tables online seeing that these Web reports will be more colorful, meaningful, and user-friendly. Developers will also benefit from these advancements as each separate component of a particular SAS/IntrNet application can be integrated into a single SAS program, one that requires very little maintenance. Ultimately, the different techniques shown provide for an application design that caters to the usability and efficiency needs of all end-users and programmers alike.

## RESOURCES

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 034-28, 2003.

Turner, Jonah P. "A Recipe for Success: Migrating SAS/AF® Applications to the Web Using SAS®, HTML, and JavaScript," *Proceedings of the Twenty-Ninth Annual SAS® Users Group International Conference*, Paper 025-29, 2004.

Turner, Jonah P. "Spicing Up SAS/IntrNet® Applications," *Proceedings of the Thirtieth Annual SAS® Users Group International Conference*, Paper 024-30, 2005.

Turner, Jonah P. "Dishing Up SAS/IntrNet® Applications – A Second Helping of a Grand Recipe," *Proceedings of the Thirty-First Annual SAS® Users Group International Conference*, Paper 233-31, 2006.



## CONTACT INFORMATION

Jonah P. Turner
International Trade Administration
1401 Constitution Ave., NW
Washington, DC 20230-0001
(202) 482-5454 or jonah.turner@mail.doc.gov