Paper 213-28

An Interactive Table for the Web Using SAS® and JavaScript

Alan Leach, Qualex Consulting Services, Inc., Norwood, OH

ABSTRACT

SAS provides a myriad of methods to produce HTML output. Add some custom JavaScript to the mixture and one can produce dynamic HTML reports. In this poster we demonstrate a web based tabular report that allows the user to collapse or expand rows of the table in order to drill into the data via a hierarchy of class variables. The HTML and JavaScript are produced using BASE SAS® and can be executed via a SAS/IntrNet® application dispatcher program or in batch to produce stored HTML pages. The JavaScript that interacts with the user makes use of recursive functions to display or hide the child rows. It is a simple way to produce interactive web pages from a SAS dataset. In this poster we will explain our web report and its functionality. We will then demonstrate the JavaScript functions that provide the user interaction. Finally we will explain how we use SAS DATA steps and PROCs to produce the HTML and JavaScript for these interactive tables. We also mention how we use cascading styles sheets to enhance the display.

INTRODUCTION

This poster will discuss an interactive web-based tabular report created to meet a client's reporting needs. It makes use of a predefined path or hierarchy of class variables that allows users to drill into lower levels. The drilling is accomplished by hiding and showing rows of the tables using JavaScript and styles. The report can be run as an application dispatcher program or as a batch program to produce saved HTML pages. Either way it provides a way to add interactivity to a simple table report using only BASE SAS. We will first explain the table and demonstrate it's functionality. Next, we will explain how the functionality is achieved using JavaScript. Finally, we will show how we used SAS to generate the HTML and JavaScript for the report.

BACKGROUND

Using SAS with complimenting web technologies allows developers to harness the power of the SAS and present information to any client with a web browser making use of the plethora of options for displaying web content.

Our client had requirements to create web-based summary tabular reports from SAS tables similar to output from PROC TABULATE. There are of course many options in SAS to do this. The rub was that these tables needed to be interactive and allow the users to drill into the data. Furthermore, for efficiency we wanted to allow them to do this without making an additional request from the server with each drill.

Our clients had a preset drill path or hierarchy of class variables to display as the rows of the report with sub totals for each hierarchy level. They also had a single class variable to display as the across dimension.

What we came up with is a simple BASE program that creates an HTML document with embedded JavaScript function calls. The called JavaScript functions are simple functions that show or hide certain rows of the table giving the illusion of expanding or collapsing the drill paths of the table.

SAMPLE REPORT

To illustrate the report we make use of the SASHELP.PRDSALES data supplied with SAS. We will be interested in the total sales (variable name ACTUAL) as our analysis variable. For the across classification we chose the MONTH column. The rows of the table will use the following path of class variables: COUNTRY -> REGION -> DIVISION. A sample PROC TABULATE using ODS HTML would look like Figure 1:

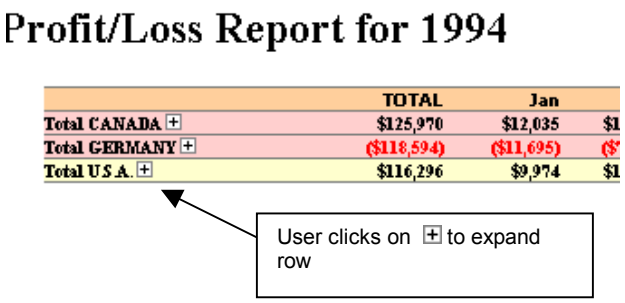
Figure 1: PROC TABULATE with ODS HTML

The SAS System

			All	Month						
				Jan	Feb	Mar	Apr	May	Jun	Jul
			Actual Sales	Actual Sales	Actual Sales	Actual Sales	Actual Sales	Actual Sales	Actual Sales	Actual Sales
			Sum	Sum	Sum	Sum	Sum	Sum	Sum	Sum
Country										
CANADA	All		125970.00	12035.00	10371.00	8894.00	12654.00	11427.00	10493.00	9605.00
	Region									
	EAST	All	63355.00	5719.00	5913.00	4685.00	6234.00	6233.00	5488.00	4247.00
		Division								
		CONSUMER	30222.00	3171.00	2949.00	1815.00	2274.00	2502.00	1918.00	2378.00
		EDUCATION	33133.00	2548.00	2964.00	2870.00	3960.00	3731.00	3570.00	1869.00
	WEST	All	62615.00	6316.00	4458.00	4209.00	6420.00	5194.00	5005.00	5358.00
		Division								

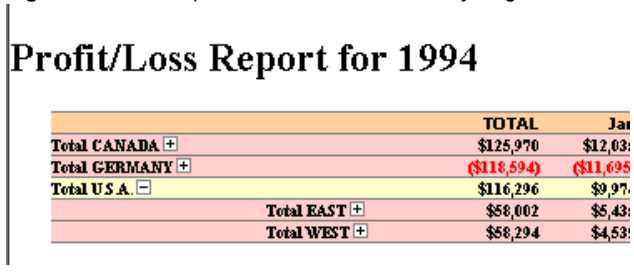
For our requirements we needed to initially display only the subtotals by COUNTRY as in Figure 2. We would allow users to expand each level of COUNTRY to examine the subtotals by REGION.

Figure 2: Initial Report Display(right side of report has been truncated)



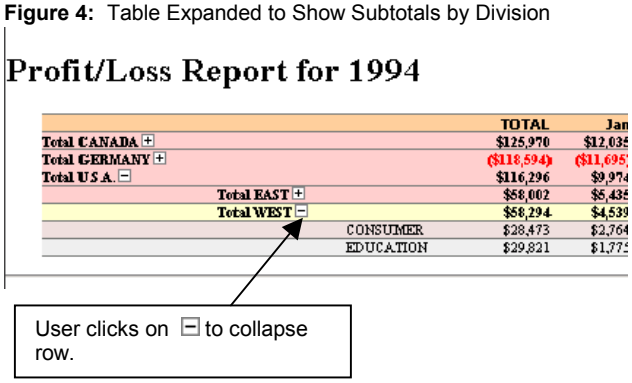
After clicking on plus sign to expand the row, the report displays as in Figure 3.

Figure 3: Table Expanded to Show Subtotals by Region



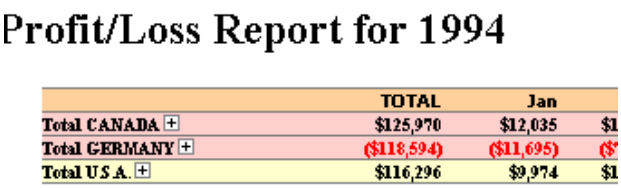
This shows the subtotals by REGION that make up the U.S.A. total. Likewise, clicking on the next to the WEST total expands the row even further as in Figure 4.

Figure 4: Table Expanded to Show Subtotals by Division



Here the subtotals by DIVISION are displayed. Notice that the images next to the row labels change to indicate the state of the row. Expanded rows can likewise be collapsed again. For example clicking on the next to the Total U.S.A label causes the table to collapse back to its origination state as in Figure 5:

Figure 5: Table Collapsed to Original State



EXPLANANTION OF HTML REPORT CODE

THE HTML PAGE

In this section we will describe the HTML and JavaScript that work to achieve the effect described above. In a later section we will describe the SAS code used to create this HTML and JavaScript.

The table displayed in the report is simply an HTML <TABLE> with table row (<TR>) and table cell tags (<TD>). Each <TR> tag uses the ID attribute to uniquely identify that row of the table as an object inside the browser window using the Document Object Model. The ID is based on the data used to generate the table. It is derived from the drill hierarchy requirements. For the above report an exert from the HTML for the table would be as follows:

```
<TABLE>
…other rows of the tables….

<!- - The USA Total Row - - >
<TR id="R_3"  noChildren=2 status="collapsed"
     style="visibility:visible"
     …other attributes…. >
    <TD> Total U.S.A.
      <! - - The little plus  or minus image  - - >
      <IMG src="/SUGI28/images/plus.gif" ID = "I_3"
        onClick="toggle('R_3')" >
   </TD>
   <TD> </TD>
   <TD> </TD>
   <TD  >  $116,296</TD>
   ……other <TD> tags for the table cells…….
</TR>

<!- - The EAST Subtotal Row for USA - - >
<TR id="R_3_1" noChildren=2  status="collapsed"
     style="visibility:hidden"
     …other attributes…. >

    <TD> </TD>
    <TD> Total EAST
       <! - - The little plus  or minus image  - - >
      <IMG src="/SUGI28/images/plus.gif" ID = "I_3_1"
       onClick="toggle('R_3_1')" >
    </TD>
    <TD> </TD>
    <TD >  $58,002</TD>
……other <TD> tags for the table cells…….
</TR>

<!- - The CONSUMER Subtotal Row for USA/EAST - - >
<TR id="R_3_1_1" noChildren=0 "
   style="visibility:hidden"

    …other attributes…. >

    <TD> </TD>
    <TD> </TD>
    <TD>CONSUMER </TD>
    <TD> ($2,750) </TD>
 ……other <TD> tags for the table cells…….
</TR>

…other rows of the tables….
</TABLE>
```

Notice the USA row has the ID = R_3 – USA is the third value (after CANADA and GERMANY) for the first class value in the hierarchy (COUNTRY). Under USA the EAST region subtotal row has the ID or R_3_1 – it is the first value of REGION for USA. We say that row R_3_1 is a child row of the parent row R_1. Similarly, row R_3_1_1 , the subtotal for DIVISION=CONSUMER, is a child row of R_3_1.

Notice also that we have added a couple custom attributes for the row. Namely the noChildren attribute that indicates how many children the row has. Also we have added a status indicator that contains the current state of the row – 'expanded' or 'collapsed'.

The rows of the table are displayed or hidden using styles. The style attribute of the <TR> tag indicates the initial display. For example, row R_3_1 is initially hidden and has style="visibility:hidden". Changing the style attribute for a specific row can hide and unhide the row in the browser.

One final item to note is the image tag <IMG> that is used in the same cell as the row label. This is the little or image that appears next to the row label. This image calls a JavaScript function when it is clicked on. This is specified in the onClick event for the image.

THE JAVASCRIPT FUNCTIONS

The toggle function called by the onClick event of the image in each parent row is the main function that does the expanding and collapsing of the table. The JavaScript code for that function is as follows:

```
function toggle(thisID){
// Shows a given rows children or hides
// a row's children and grandchildren
```

```
 // Get the reference to the Object
 var objectID = document.getElementById(thisID) ;

  // Get its current status
 var status   = objectID.status ;

 // If the row is current expanded then collapse it by
 // hiding the children .

 if ( status == "expanded" ) {
     hideChildren ( thisID ) ;
    }

 // likewise we will expand a collapsed row by
 // showing its children

 else if ( status == "collapsed" ) {
     showChildren ( thisID ) ;
     }
} // toggle function
```

Depending on the current status of the row the toggle function either calls the showChildren or hideChildren functions. These functions change the style visibility attribute for all the child rows. For example the showChildren function is as follows:

```
function showChildren ( rowID ) {
// Displays all the children for given parent

 var childName  ;

  // Get the Object reference for the passed Row
 var parentObject = document.getElementById ( rowID ) ;

  // Get the number of children for the row
 var noChildren   = parentObject.noChildren ;

 // Build the name of the image associated with the row
 var imgName = "I_" + rowID.substr( 2,rowID.length );

  // Get the Object reference for the image associated with the
row
 var imgObject = document.getElementById( imgName ) ;

  var i ;

 // If there are no children then we are done

 if  ( noChildren == 0 ) {
    return
    }

 // Otherwise unhide each child row
 else
  {
     // For each child...

     for ( i = 1; i <= noChildren ; i++ )

     {

       // Build the name of the child...

       childName = rowID + "_" + i  ;

        // Change the visibiltity of the child

   document.getElementById(childName).style.visibility="visible";

     }

     // Change the status of the parent to expanded
     parentObject.status = "expanded";

     // Change image to indicate the row is expanded....
     imgObject.src = "/SUGI28/images/minus.gif";

   }

} // showChildren function
```

In this function we make use of the consistent row IDs that we have created to build the name of the child rows. With the noChildren attribute and the ID of the parent row we can build the name if each child row in succession and use that to change the style of each child row. Note also that we change the image associated with the parent row to the minus.gif image to indicate that row is expanded.

When the user wants to collapse the table rows, i.e., hide the child rows, when need a slightly different function, the hideChildren function. Since a given row may grandchildren, i.e. its children may have children, we also have to call this function for the child row. In other words, the hideChildren function may have to call itself. This is the simple elegance of recursion. The hideChildren function is as follows:

```
function hideChildren ( rowID ) {
// Hides all the children for given parent

 var childName  ;

 // Get the Object reference for the passed Row
 var parentObject = document.getElementById ( rowID ) ;

 // Build the name of the image associated with the row
 var imgName = "I_" + rowID.substr( 2,rowID.length );

 // Get the Object reference for the image associated with the row
 var imgObject = document.getElementById( imgName ) ;

 var i = 1;

 // If there are no children then we are done
 if  ( noChildren == 0 ) {
    return
   }

 // Otherwise unhide each child row

 else {

   // For each child...

   for ( i = 1; i <= noChildren ; i++ )

    {

      // Build the name of the child...
```

```
      childName = rowID + "_" + i  ;

      // Call the hideChildren passing the current child ID as a
      // parnet ID.  This will hide all the children and grandchildren
      // for the row.

      hideChildren ( childName ) ;

      // Hide the current row

      hideObject ( document.getElementById ( childName ) ) ;

     }

    // Change the status of the parent to collapsed

   parentObject.status = "collapsed";

   // Change image to indicate the row is expanded....

   imgObject.src = "/SUGI28/images/plus.gif";

   }

} // hideChildren function
```

Typically these JavaScript functions are defined in a .js file that is stored somewhere in a sub directory of the web server and are referenced by a <SCRIPT> tag in the header of the HTML page (see below).

SO WHERE DOES SAS COME IN?

The HTML Table that displays the data and contains the function call the interact with the user is generated by SAS. Using DATA _NULL_ in BASE SAS the HTML code is generated from the source data. The key part that SAS plays, apart from summarizing the data, is in creating the ID attributes for the <TR> tags. Using a simple PROC MEANS and DATASTEPs in SAS we are able to dynamically generate the <TR> tags for the table based on the data at hand.

First , we summarize the source data to the levels that we need:

```
/***********************************/
/* Summarize the source data and   */
/* output to a dataset to          */
/* be used in reporting.  Note     */
/* the use the types statement     */
/* to only provide the summary     */
/* levels needed.                  */
/***********************************/


proc means data = sasdl.prdsale sum   noprint
chartype completeypes;

   class COUNTRY REGION DIVISION MONTH;

   type  COUNTRY * REGION * DIVISION * MONTH
         COUNTRY * REGION            * MONTH
         COUNTRY                     * MONTH
         COUNTRY * REGION * DIVISION
         COUNTRY * REGION
        COUNTRY
         ;

   var actual ;

   output out   =  summary
    ( Keep =   COUNTRY REGION DIVISION
                MONTH _TYPE_ ACTUAL )
          sum ( actual ) = actual
   ;
run;
```

Within this summarization we need to count the number of child levels or rows for each class variable. For example, within each value of COUNTRY, we need to determine how many unique values of REGION exist. Also, within each value of REGION we need to know how many unique values of DIVISION exist. We accomplish this with another PROC MEANS to do the counts.

```
/***************************************/
/* Here we need to get the number of   */
/* child class variable values within  */
/* each parent level.For example       */
/* within COUNTRY = 'GERMANY' , there  */
/* are two REGIONS, 'EAST' and         */
/* 'WEST'.  Thus COUNTRY = 'GERMANY'   */
/* has two children.                   */
/***************************************/

PROC MEANS n noprint  nway data = summary;
    where _TYPE_ = "1100" ;
    class COUNTRY   ;
    output out = COUNTRY_COUNTS
     ( keep = COUNTRY _STAT_ _FREQ_
       where = ( _STAT_ = "N" )
     /* All we need are the counts */
    rename = ( _FREQ_ = noChildren ) ) ;
run;


PROC MEANS n  noprint nway data = summary;
  where _TYPE_ = "1110" ;
  class COUNTRY REGION  ;
  output out = REGION_COUNTS
  ( keep = COUNTRY REGION _STAT_ _FREQ_
    where = ( _STAT_ = "N" )
   /* All we need are the counts */
     rename = ( _FREQ_ = noChildren ) ) ;

run;

/***************************************/
/* Merge these child counts back into  */
/* data to draw the table.             */
/***************************************/


data summary ; merge summary  country_counts
 ( drop = _STAT_ )  ;
 by country ;
run;


data summary ; merge summary  region_counts
  ( drop = _STAT_ ) ;
  by country region ;
run;
```

(Since DIVISION is at the bottom of our hierarchy it will have no children.)

At this point we have all the information we need to create the report. DATA _NULL_ processing will create the opening HTML tags as well as the <HEADER> tags that reference our JavaScript file using a <SCRIPT> tag. We will also create the opening <BODY> tag:

```
data _null_;
      file _webout ;
      put '<html>';
      put '<head>';
      put '<title>SUGI 28 </title>';
      put '<link rel="stylesheet"
```

```
           href="/SUGI28/styles/SUGI28.css">';


/*******************************************/
/* Note the reference to our .JS files that  */
/* contains the functons definitions that    */
/* the program.                              */
/*******************************************/
      put '<script language="javascript"
src="/SUGI28/javascript/sastable.js"></script>';

      put '</head>';
      put '<body BGCOLOR="#FFFFFF"
            LEFTMARGIN="0" TOPMARGIN="0"
            MARGINHEIGHT="0" MARGINWIDTH="0">';
```

Once our opening tags for the HTML page has been output we will write the opening <TABLE> tags output the column headers. Once this is done we use our summary dataset to create the data table. The general procedure is as follows: For each new class value in our hierarchy we open a row of the table(<TR>) and output a label for the row in a table cell(<TD>). Each record in the dataset corresponds to a different MONTH and these MONTHs will be the table columns (<TD>). For the last value of a variable in the hierarchy we will close the table row (<TR>).

This is the code the begins to create the table and opens the subtotal rows for COUNTRY, the first variable in the drill hierarchy:

```
data _null_;

  set summary;

  by COUNTRY REGION DIVISION ;  /* Use BY in
              order to use first. processin */

  file _webout ;   /* Output to _WEBOUT to send
                      back to the browser or
     another fileref to create permanent file */

  length row_id $ 40 ;

  retain c_COUNTRY c_REGION c_DIVISION 0;

/*******************************************/
/* Open the table row for the first value     */
/* in the first variable in the report        */
/* hierarchy . COUNTRY .                      */
/*******************************************/

   if first.COUNTRY then do ;

       c_COUNTRY + 1 ;  /* Increatment the
counter keep track of the number of levels for
COUNTRY */


/*******************************************/
/* Create the ROW_ID and IMG_ID based on the */
/* level number.  These will be used by the  */
/* Javascript to identify the                */
/* row and image on that row.                */
/*******************************************/

row_ID = "R_" || trim( left ( put ( C_COUNTRY,
                                    3. ) ) );

img_ID = "I_" || substr ( row_ID , 3 ) ;

/*******************************************/
/* Output the HTML for the table row.  Note   */
/* that we output the number of               */
/* children as an attribute and well as the   */
/* status of the row.                         */
/*******************************************/
```

```
put '<TR id="' row_id   +(-1)
    '" noChildren=' noChildren '
    style="visibility:visible"
    status="collapsed" class ="subTotalrow">';

       /*************************/
       /* Open the data cell for */
       /* the label.             */
       /*************************/

       put "<TD> Total " COUNTRY  ;


/*******************************************/
/* If there children we display a plus sign   */
/* image.  The onClick event calls the toggle */
/* function for the row.  This will swap the  */
/* status of the of the current row.          */
/*******************************************/

 if noChildren > 0 then do ;
    put '    <IMG src="/SUGI28/images/plus.gif"
    ID = "' img_ID  +(-1) '"
      onClick="toggle(' "'" row_ID  +(-1) "'" '
      ')" >';

  end ;
  else do;

    put ' <IMG src="/SUGI28/images/leaf.gif" >';

  end;

      /*************************/
      /* Close the data cell     */
      /* and output two spacer   */
      /* cells .                 */
      /*************************/

       put "</TD>" ;
       put "<TD> </TD>"  ;
       put "<TD> </TD>"  ;

       /*********************************/
        /* Reset the counter for the next  */
       /* Class variable in the hierachy  */
       /*********************************/

           C_REGION = 0 ;
 end;
```

The steps for opening the table rows for REGION and DIVISION are similar. When FIRST.REGION and FIRST.COUNTRY are true there would be a block of code similar to that above for FIRST.COUNTRY. The key difference is that the row_ID is built differently for these levels of the hierarchy. For example, for REGION the row_ID is based on the COUNTRY and REGION as follows:

```
row_ID = "R_" ||
         trim( left ( put ( C_COUNTRY, 3. )))
         || "_" ||
         trim ( left ( put ( C_REGION, 3. )))
          ;
```

Likewise the row_ID for the DIVISION subtotals will be based on the COUNTRY, REGION and DIVISION:

```
row_ID = "R_" ||
         trim( left ( put ( C_COUNTRY, 3. )))
         || "_" ||
         trim ( left ( put ( C_REGION, 3. )))
         || "_" ||
         trim ( left ( put ( C_DIVISION, 3. )))
          ;
```

After the table rows are opened we output a table cell for each MONTH by outputting each record in the dataset:

```
/*******************************************/
/* Output the data cell for the table using  */
/* the appropriate format.                   */
/*******************************************/

   put  "<TD style='text-align:right'>"
        actual account.   "</TD>" @ ;
```

When the last value of DIVISION is read we close the table row:

```
/*******************************************/
/* For the last level of the last variable in */
/* the hierarchy we close the HTML table row  */
/*******************************************/


if last.DIVISION and division ne " "  then do;

         put "</TR>";
end;
```

Once the table is drawn a DATA _NULL_ is used to close the table and the HTML page.

This completes the report. The row IDs built by SAS are passed to the toggle function in each onClick event to do the hiding and showing of the rows.

ADDITIONAL STYLES ITEMS USED

In addition to the interactivity of the table we needed a way needed to display negative numbers in red in our report for quick reference by the users. To do this we created formats with embedded HTML. (Note that HTML 3.2 allows these specifications while this is deprecated in HTML 4.0 in favor of cascading style sheets.) One such format we used is as follows:

```
/*********************************/
/* ($xxx,xxx) w/ HTML            */
/*********************************/
proc format;
  picture account
    low - < 0     = '000,000,000) </FONT>'
                  (prefix='<FONT COLOR=RED> ($')
    0             = '9'
                  ( prefix = '$' )
    0 < - high =  '000,000,000'
                  (prefix='$')
              . = ' '
    ;
run;
```

Furthermore, we make use of cascading style sheets to define the look and feel of the content of the HTML page. Typically, these cascading styles sheets exist in a .css file somewhere out on the web server and are referenced in the <HEAD> section of the HTML document. They define certain style classes that are used in the HTML page. For example, we use the subtotalRow class that defines the look of table rows for our subtotals. In an external file we have a definition for that class as follows:

```
.subtotalRow
{
   BACKGROUND-COLOR: #ffcccc;
   FONT-SIZE: 7pt;
   FONT-STYLE: normal;
   FONT-VARIANT: normal;
   FONT-WEIGHT: bold;
   LINE-HEIGHT: normal
}
```

Sometimes we need to override specific style properties using the STYLE= tag in the HTML as you will notice in the code above.

FURTHER ENHANCEMENTS

Our example, purposely made simple to illustrate the general techniques, can be used to create more complicated reports. For example, most reports would show subtotals above the grand totals instead of below as in our example. To change this additional data steps and a changes in the DATA _NULL_ that create the table would be needed. (The code to do this is available upon request from the author).

Also, in practice report tables may be too wide to fit across on a browser screen. Thus, several tables, each a browser screen wide, stacked on top of each other on different 'pages', might be used. Not only would the SAS code to create these table be different but the JavaScript to expand /collapse the rows would need to take this additional dimension of page.

CONCLUSIONS

In this paper we have illustrated just one example of how SAS can be used with other technologies to produce web based reports. Using JavaScript and styles the HTML output easily produced in SAS can be customized to meet a wide variety of requirements allowing one to exploit the power of SAS for the web.

REFERENCES

Goodman, Danny. 2001. JavaScript Bible, Gold Edition. Hungry Minds, Inc., New York, NY.

Musciano, Chuck and Kennedy, Bill. 1998.HTML – The Definitive Guide, Third Edition. O'Reilly and Associates, Sebastopol, CA.

Lie, Håkon, Wium and Bos, Burt. 1999 Cascading Styles Sheets – Designing for the Web, Second Edition. Addison-Wesly, London.

SAS Instituite Inc. "SAS/IntrNet SoftWare Documentation Index". <http://www.sas.com/rnd/web/intrnet/sitemap.html>

TRADEMARK CITATION

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Also the complete source code needed to create the reports mentioned in the paper is available upon request from the author. Contact the author at:

Alan Leach
Qualex Consulting Services, Inc.
2217 Cleneay Ave
Norwood, OH 45212
Work: 513.731.8831
Email: Alan.Leach@qlx.com
Web: www.qlx.com