Paper 22

Data Tables: Highlighting Techniques

Christopher A. Roper, Qualex Consulting Services, Inc.

Abstract

Data tables are arguably one of the best looking widgets in SAS®. This makes them a popular choice for replacing the venerable Extended Table widget as a selection list, or front end to a data set. The model/viewer paradigm is an extremely flexible tool that gives the Data Table, and it's parent class the Table Editor, an extensive utility across a broad range of applications. Combined with the abundance of methods available to the Data Table class, this widget can add a very nice touch of polish to an application, giving it the professional, finished appearance a client demands and deserves. This paper will illustrate five scenarios for using highlighting to make the Data Table object more attractive and functional for your information consumers.

Introduction

Highlighting rows, columns, or individual cells in a Data Table is a powerful method to emphasize important information, or remind the end-user of previous selections. There are two general input triggers available to signify when highlighting should occur. The first is a data driven trigger, some predetermined value of a variable that will cause highlighting to occur when it is observed. Data driven highlighting may be column based or row based. The other trigger is a user initiated event, such as the selecting of a cell. User initiated highlighting may be column base, row based, or cell based. This paper will present techniques that will accomplish each of these five highlighting tasks to illustrate how highlighting can be used in the Data Table widget to make an application easier to use and more visually impressive.

The Default Attributes

In order to make the examples as simple as possible and enhance portability, each technique illustrated will use the following as defaults. The data set will be "SASHELP.IAWFIN". The frame entry will reside in a catalog named DATATBL.HIGHLITE, and will contain one widget, named "TABLE", which is an instance of Data Table class with all of the object attributes left to the original default settings, with the exception of the model's SCL entry. For some examples, the model's SCL entry will be used to accomplish the highlighting, and therefore each of those examples will have a different SCL entry for the table's model. Highlighting is defined as the controlling of the background color of cells to make them standout from the other cells of the table. The foreground color of the cells may also be changed, to keep the text in contrast with the background.

Data Driven Highlighting

Data driven highlighting is the controlling of the background color of table cells based on the value of a cell. Since the Data Table class may have only two dimensions, i.e. a row dimension and a column dimension, this section will provide a technique for highlighting the individual cells of a column, and a technique for highlighting the individual cells of a row.

Highlighting A Column Dimension

Highlighting the individual cells of a column dimension is the simplest of the highlighting tasks to accomplish. For this illustration, the values of the variable VARIANCE from the date set SASHELP.IAWFIN, will be highlighted according to the table below.

Table 1

Variance	Highlight
-100 to -50	Orange
- 49 to -25	Yellow
- 24 to 25	None (White)
26 to 50	Cyan
51 to 100	Magenta

With the advent of version 6.12 came a new method to make controlling the appearance of data tables much easier to do. The _SET_VIEWER_ATTRIBUTE_ method was designed for just this purpose. Entering the name of an SCL entry in the SCL ENTRY field of the data table widget's object attributes, creates the model's SCL entry. This entry may be edited just as any other SCL entry, or by selecting the TABLE,EDIT

SCL options from the popmenu presented by a right-mouse click on the data table object. Simply put, the INIT section of the data table model's SCL executes once for each displayed row in the data table. Therefore, to individually highlight the cells of a column dimension based upon the values in the cell, simply compare the current column value with a given threshold, and perform the highlighting as necessary. Example 1 is the described model's SCL.

Example 1

```
INIT:  /*  Model's INIT SCL  */
  if variance le -50 then
  do;  /*  Highlighting for LE  -50  */
    highlight = 'ORANGE';
    foreground = 'WHITE';
  end;  /*  Highlighting for LE  -50  */

  else if variance le -25 then
  do;  /*  Highlighting for -49 to -25  */
    highlight = 'YELLOW';
    foreground = 'BLACK';
  end;  /*  Highlighting for -49  to -25  */

  else if variance le 25 then
  do;  /*  Highlighting for -24 to 25  */
    highlight = 'WHITE';
    foreground = 'BLACK';
  end;  /*  Highlighting for -24 to 25  */

  else if variance le 50 then
  do;  /*  Highlighting for 26 to 50  */
    highlight = 'CYAN';
    foreground = 'BLACK';
  end;  /*  Highlighting for 26 to 50  */

  else
  do;  /*  Highlighting for GT 50  */
    highlight = 'MAGENTA';
    foreground = 'WHITE';
  end;  /*  Highlighting for GT 50  */

/***********************************/
/*    Apply the highlight and text color    */
/***********************************/
  call send(_viewer_,
       '_SET_VIEWER_ATTRIBUTE_',
       'VARIANCE','BCOLOR',highlight);

call send(_viewer_,
'_SET_VIEWER_ATTRIBUTE_',
       'VARIANCE','FCOLOR',foreground);

return;  /* Model's  INIT SCL  */
```

Highlighting A Row Dimension

Highlighting the cells of a row dimension is often useful if the rows have some kind of positional authority. In other words, the order of the records in the table is important, and constant. For example, transpose the data in SASHELP.IAWFIN so that the resultant dataset contains five records or rows, with each row corresponding to the previous columns HUB, AREA, BUDGET, ACTUAL, and VARIANCE. Example 2 details the code to accomplish this:

Example 2

```
Proc transpose data = SASHELP.IAWFIN
                out = ROWDATA(drop=_label_);
where HUB = 'Frankfurt' and
      AREA = 'Equipment';
var HUB AREA BUDGET ACTUAL VARIANCE;
run;
```

In the object attributes window, the transposed data set is attached to the data table. The highlighting will be performed according to Table 2:

Table 2

Variance	Highlight
Greater than 50	Orange
21 to 50	Yellow
Less than 21	Green

Example 3 is the model's SCL code required to accomplish the highlighting. This code is very similar to the code for example 1. The only difference is in making the highlighting code conditional, based on the value of _NAME_, created by the transpose procedure. Example 3 shows how the row VARIANCE, (the row in which the value of _NAME_ equals "VARIANCE") drives the highlighting of the data table.

Example 3

```
length col_val $16 column $8;

INIT:  /*  Model's SCL INIT  */
/***********************************/
/*   Highlight only the VARIANCE row    */
/***********************************/
  If _NAME_ = 'VARIANCE' then
  do; /*  Determine the row to highlight */

     dsid = open('work.rowdata','I');
```

```
/*********************************/
/*   Highlight each column in this row    */
/*********************************/
  do i=1 to attrn(dsid,'NVARS');

    /*************************/
    /*  Determine column name  */
    /*************************/
    column = varname(dsid,i);
    call send(_self_,
              '_GET_COLUMN_TEXT_',
              column,col_val);
    col_val = compress(col_val);

    /*************************/
   /*   Set the highlighting colors */
   /*************************/
    if col_val = 'VARIANCE' then
    do;  /*  No Highlight for Row Labels  */
      highlight = 'BACKGROUND';
      text = 'FOREGROUND';
    end; /*  No Highlight for Row Labels  */

    else if col_val gt '50' then
    do; /*  Highlight for GT 50  */
      highlight = 'ORANGE';
      text = 'WHITE';
    end; /*  Highlight for GT 50  */

    else if col_val gt '20' then
    do;  /*  Highlight for 21 to 50  */
      highlight = 'YELLOW';
      text = 'BLACK';
    end;  /*  Highlight for 21 to 50  */
    else
    do;  /*  Highlight for LT 21  */
      highlight = 'GREEN';
      text = 'WHITE';
    end;  /*  Highlight for LT 21  */

/*********************************/
/*     Set the cell's highlight color      */
/*********************************/
    call send(_viewer_,
         '_SET_VIEWER_ATTRIBUTE_',
         column,'BCOLOR',highlight);

/*********************************/
/*       Set the cell's text color          */
/*********************************/
     call send(_viewer_,
          _SET_VIEWER_ATTRIBUTE_',
           column, 'FCOLOR',text);
  end;/* do i=1 to attrn(dsid,'NVARS') */

/*********************************/
/*              Clean Up                    */
/*********************************/
  if dsid then rc = close(dsid);
  rc = rc;

end;  /*  If _NAME_='VARIANCE'  */

return;  /*  Model's SCL INIT  */
```

End-User Initiated Highlighting

End-user initiated highlighting is the setting of the background color for a cell or group of cells based on the end-user causing an event to occur in the table. Typically this is the result of a single or double click on a cell within the table. The remainder of this paper will deal with three possible scenarios resulting from the selection of a data table cell; highlighting the corresponding column, highlighting the corresponding row, and highlighting just the selected cell.

Highlighting A Column

This example will demonstrate how to highlight an entire column when any cell within that column has been selected. This will be done without overwriting any methods nor will the model's SCL entry be required to perform any tasks. In short, all the processing required for this example occurs in the labeled section for the data table in the frame's SCL entry. The first task is to determine which column has been selected. Then, the decision of whether to highlight or not must be made. Toggling the background color from highlighted to the default background is done using the method _GET_COLUMN_ATTRIBUTE_ to pass the current color, which is then reassigned and applied to the data table by setting the data background color value in the list (in this case named SEL_COLS), used by the _SET_COLUMN_ATTRIBUTES_ method.

Example 4

```
length colname $8 color $16;

INIT:   /* Frame's INIT  */

 call send(_frame_,'_GET_WIDGET_',
                   'TABLE',t_id);

 call send(t_id,'_SET_DATASET_',
                 'SASHELP.IAWFIN");
```

```
return;  /*  Frame's INIT  */

TABLE:
/*********************************/
/* Highlight/Unhighlight Select Column */
/*********************************/
 lsidrow=makelist();  /* Selected Row */
 lsidcol=makelist();  /* Selected Column */
 sel_cols = makelist(); /* Column Attrs */



/*********************************/
/*     Get the selected column number     */
/*********************************/
  call send(t_id,'_GET_ACTIVE_CELL_',
                  lsidrow,lsidcol);
  col = getitemn(lsidcol,1);

/*********************************/
/*     Get the selected column name       */
/*********************************/
  call send(t_id,
'_GET_DISPLAYED_COLUMN_NAME_',
                  col,colname);

/*********************************/
/*      Get the current column color       */
/*********************************/
  call send(t_id,
          '_GET_COLUMN_ATTRIBUTE_',
          colname,'DBCOLOR',color);

/*********************************/
/*        Toggle the highlight color         */
/*********************************/
 select(color);
   when('BACKGROUND')
                  color='YELLOW';
   otherwise
                  color = 'BACKGROUND';
 end;    /* Select Color */

/*********************************/
/*     Set the selected column highlight    */
/*********************************/
  sel_cols=setnitemc(sel_cols,colname,
                     'NAME');
  sel_cols=setnitemc(sel_cols, color,
                     'DBCOLOR');

/*********************************/
/* Apply the selected column highlight */
/*********************************/
  call send(t_id,
          '_SET_COLUMN_ATTRIBUTES_',
          sel_cols);

/*********************************/
/*                Clean up                */
/*********************************/
 if lsidrow then lsidrow=dellist(lsidrow);
 if lsidcol then lsidcol=dellist(lsidcol);
 if sel_cols then sel_cols=dellist(sel_cols);

/*********************************/
/*   Force a refresh of the model/viewer */
/*********************************/
 call send(t_id,'_UPDATE_');

return;   /*  TABLE  */
```

Highlighting A Row

Highlighting rows in a data table often means overwriting the _GET_DATA_ method. While that technique is still valid and popular, this example will demonstrate an alternative technique, one that does not require overwriting any existing methods. This method will use the frame's SCL entry (Example 5) to track which rows have been selected and deselected by putting the row numbers of the selected rows in an SCL list, and removing the row numbers of deselected rows from that list. The model's SCL entry (Example 6) will then compare the currently active row with the row numbers in the list and highlight accordingly. The local environment list will be used as a vehicle to pass the row numbers, contained in the list LSIDSEL, between the two SCL entries.

Example 5 (Frame's SCL)

```
length color $16;

INIT:

 call send(_frame_,'_GET_WIDGET_',
                   'TABLE',t_id);

 lsidlocl = envlist('L');
 lsidsel = makelist();
 lsidlocl = insertl(lsidlocl,lsidsel,
                   -1,'Sel_Rows');

return;  /*  Frame's INIT  */
TABLE:  /*  Data Table Widget  */

/*********************************/
/*       Determine the selected row         */
```

```
/*********************************/
 lsidrow = makelist();
 lsidcol = makelist();

 call send(t_id,'_GET_ACTIVE_CELL_',
                lsidrow,lsidcol);
 row = getitemn(lsidrow,1);

/*********************************/
/*  Check if the row had been selected  */
/*  earlier, remove it if so, add it if not */
/*********************************/
 selected = searchn(lsidsel,row);

 if selected then
    lsidsel = delitem(lsidsel, selected);
 else lsidsel = insertn(lsidsel,row);

/*********************************/
/*  Remove the box around the cell,      */
/*  Refresh the data table               */
/*********************************/
 call send(t_id,'_CLEAR_ACTIVE_CELL_');

/*********************************/
/*              Clean up                 */
/*********************************/
 if lsidrow then lsidrow = dellist(lsidrow);
 if lsidcol then lsidcol = dellist(lsidcol);

return;  /*  TABLE  */

TERM:
   if lsidsel the lsidsel = dellist(lsidsel);
return;  /*  TERM  */
```

After the frame's INIT section has run, the model's SCL entry INIT section will execute once for each displayed row of the table. Then, when a row is selected the model's SCL INIT section will execute for the selected row *before the labeled section for the table runs.* This last part is important to remember, and is the reason the method _CLEAR_ACTIVE_CELL_ is called near the end of the data table's labeled section, after the list LSIDSEL has been updated. This method causes the model's SCL to run again, so that the updated list of selected rows may be processed by the model's SCL. The model's SCL compares the current row to the list of selected rows, and highlights accordingly using the _SET_VIEWER_ATTRIBUTE_ method available in version 6.12.

Example 6 (Model's SCL)

```
INIT:   /*  Model's INIT section  */

/*********************************/
/*  Get the list of selected rows from the  */
/*  local environment list                  */
/*********************************/
  if lsidlocl = . then
  do;  /*  List has not been retrieved  */
    lsidlocl = envlist('L');
    lsidsel = getniteml(lsidlocl,'Sel_Rows');
  end;  /*  List has not been retrieved  */

/*********************************/
/*  Determine the current row number     */
/*********************************/
  call send(_viewer_,
'_GET_CURRENT_ROW_NUMBER_',currow);

/*********************************/
/*  Check if this row has been selected     */
/*  if so, highlight, else remove highlight */
/*********************************/
  selected = searchn(lsidsel,currow);

  if selected then
    call send(_viewer_,
              '_SET_VIEWER_ATTRIBUTE_',
              '_ALL_', 'BCOLOR','YELLOW');

  else
    call send(_viewer_,
            '_SET_VIEWER_ATTRIBUTE_',
    '_ALL_','BCOLOR','BACKGROUND');

return;  /*  Model's SCL INIT   */
```

Highlighting A Cell

Highlighting a cell in a data table is very similar to highlighting a row, essentially the technique is the same. But instead of highlighting the entire row, just the selected column of the row is highlighted. This example will use the frame's SCL entry (Example 7) to track which row and column combinations have been selected and deselected.

This is accomplished by creating separate lists for each selected row, giving that list the name of the selected row number, and filling the list with the names of the selected columns in the row. These lists are then inserted into a parent list (LSIDROWS), which can then be queried using the NAMEDITEM SCL function to determine if the current row has been selected previously.

If the current row has been previously selected, it will be a sublist in LSIDROWS and that sublist can then be queried for the selected column name. If the

current column name is in the row sublist, then the active data table cell is being deselected, and the column name must be removed from the row sublist. If the current row has not been previously selected, then a new named list is created for that row. It is given the row number as it's name, the selected column name is inserted in the list, and the list is made a sublist of LSIDROWS.
The model's SCL entry (Example 8) will then compare the currently active row number with the named items in the list LSIDROWS and when a match is found, highlight the column names found in the sublist. The local environment list will again be used as a vehicle to pass the sublists contained in the list LSIDROWS between the two SCL entries.

Example 7 (Frame's SCL)

```
length colname $8;

INIT:

/*************************************/
/*  Determine the id of the Data Table Object  */
/*************************************/
  call send(_frame_,
            '_GET_WIDGET_','TABLE',t_id);

/*************************************/
/*  Create the parent list for the selected rows */
/*  and insert it into the local environment list */
/*************************************/

 lsidlocl = envlist('L');
 lsidrows = makelist();
 lsidlocl = insertl(lsidlocl,lsidrows,-1,'Sel_Rows');

return;


TABLE:  /*  Data Table Object  */

/*************************************/
/*  Initiate work lists for row and column ids */
/*************************************/

 lsidrow = makelist();
 lsidcol = makelist();

/*************************************/
/*       Determine the selected cell address      */
/*************************************/

 call send(t_id,'_GET_ACTIVE_CELL_',
                  lsidrow,lsidcol);
 col = getitemn(lsidcol,1);
 row = getitemn(lsidrow,1);

/*************************************/
/*       Retrieve the selected column name       */
/*************************************/
 call send(t_id,
       '_GET_DISPLAYED_COLUMN_NAME_',
       col,colname);

/*************************************/
/* Determine if the cell has been previously */
/* selected by checking the row (a named     */
/* item in the list LSIDROWS)  and then the */
/* column name (in the row sublist)          */
/*************************************/

 if nameditem(lsidrows,quote(row)) then
 do;   /*  Row previously selected  */

    /*********************************/
    /*  Manage the deselecting of the last   */
    /*  selected column in the current row  */
    /*********************************/

    /*********************************/
    /*      Get the list for the current row     */
    /*********************************/
    lsidprow = getniteml(lsidrows,quote(row));

    /*********************************/
    /* If the current column is the only one  */
   /* in this sublist, delete the sublist         */
   /*********************************/
    if listlen(lsidprow) eq 1 and
      colname = getitemc(lsidprow) then
    do;  /*  Deselected the last column    */

      lsidprow = dellist(lsidprow);
      lsidrows = delnitem(lsidrows,
                          quote(row));

    end;  /*  Deselect the last column   */
    else
    do;   /*  Column Names in the sublist  */

      /*******************************/
      /* If the current column name is in */
     /* the sublist, remove it.  Otherwise */
     /* add it to the sublist                */
     /*******************************/
       p_col = searchc(lsidprow,colname);
      if p_col then
        lsidprow = delitem(lsidprow,p_col);
     else lsidprow =
        insertc(lsidprow,colname);
```

```
    end;  /*  Column Names in the sublist  */

 end;  /*  Row previously selected  */
/*************************************/
/*  For new row selections, create a sublist  */
/*  name it the current row number, and       */
/*  insert the name of the current column     */
/*************************************/
 else
 do;   /*  New Row selection  */

   lsidprow = makelist();
   lsidrows = insertl(lsidrows,
                      lsidprow,-1,quote(row));
   lsidprow = insertc(lsidprow,colname);

 end;  /*  New Row Selection  */

/*********************************/
/*  Remove the box around the cell,      */
/*  Refresh the data table               */
/*********************************/

 call send(t_id,'_CLEAR_ACTIVE_CELL_');

/*********************************/
/*                Clean up               */
/*********************************/

 if lsidrow then lsidrow = dellist(lsidrow);
 if lsidcol then lsidcol = dellist(lsidcol);

return;  /*  TABLE  */


TERM:

 if lsidrows then lsidrows = dellist(lsidrows,'Y');
 lsidlocl = delnitem(lsidlocl,'Sel_Rows');

return;  /*  TERM  */
```

Now that the frame's TABLE section has run and created the sublists identifying the cells in the data table that are to be highlighted, it is the task of the data table model's SCL to apply the highlighting to those cells. As in the frame's SCL, the model's SCL for highlight cells is very similar to row highlighting. Using the row highlighting model's SCL as a basis, example 8 shows how to check the sublists created by the frame's SCL to identify the cells that are to be highlighted. The sublists are made available to the model's SCL via the local environment list. The rows containing cells to be highlighted are identified by the NAMEDITEM function, and the columns to be highlighted in those rows are identified by the SEARCHC function. Given this information, the specified cell is identified and the _SET_VIEWER_ATTRIBUTES_ method can be used to apply the highlight.

Example 8 (Model's SCL)

```
INIT:

/*********************************/
/*  Get the list of selected rows from the  */
/*  local environment list                  */
/*********************************/

  if lsidlocl = . then
  do;   /*  Local Environment List Missing  */

    lsidlocl = envlist('L');
    lsidrows = getniteml(lsidlocl,'Sel_Rows');

  end;  /*  Local Environment List Missing  */

/*********************************/
/*    Determine the current row number    */
/*********************************/

  call send(_viewer_,
         '_GET_CURRENT_ROW_NUMBER_',
         currow);

/*************************************/
/*  Check if the current row has selected      */
/*  columns in it                              */
/*************************************/

  sel_row = nameditem(lsidrows,
                      quote(currow));


/*************************************/
/*  Highlight the selected columns in the      */
/*  current row                                */
/*************************************/

  if sel_row then
  do;   /*  Col(s) in this row were selected  */

     /*****************************/
     /*  Get the list of selected columns  */
     /*****************************/

     lsidrow = getiteml(lsidrows,sel_row);
```

```
    /*********************************/
    /*  Highlight each selected column      */
    /*********************************/

     do i = 1 to listlen(lsidrow);

      /*****************************/
      /*  Get the name of the column     */
      /*****************************/

      curcol = getitemc(lsidrow,i);

     /*************************/
     /*  Apply the highlighting       */
     /*************************/

      call send(_viewer_,
                  '_SET_VIEWER_ATTRIBUTE_',
                  curcol,'BCOLOR','YELLOW');

    end;  /*  do i = 1 to listlen(lsidrow)  */

  end;  /*  Col(s) in this row were selected  */

/*************************************/
/*  Apply the default background color to the  */
/*  unselected cells                           */
/*************************************/

  else
    call send(_viewer_,
                '_SET_VIEWER_ATTRIBUTE_',
           '_ALL_','BCOLOR','BACKGROUND');

return;  /*  Model's INIT  */
```

Conclusion

This paper is presented to provide techniques for taking advantage of the new method _SET_VIEWER_ATTRIBUTES_ available in version 6.12 of the SAS® system to perform highlighting in the Data Table widget. These techniques demonstrate that it is no longer necessary to overwrite any existing methods (i.e. _GET_DATA_) to accomplish the highlighting, thus eliminating any difficulties associated with overwriting methods. Additionally, the online help facility of SAS provides all the documentation necessary to adapt the code from these examples to specific applications.

Author Contact Information

Christopher A. Roper

Qualex Consulting Services, Inc.
Rt. 2 Box 224C
Hillsville, Va. 24343
(540) 398-3757
CHRIS.ROPER@QLX.COM

Acknowledgments