Overview

The Table Editor class is part of a collection of classes that combine to create the Data Form and Data Table classes. These classes work together to enable you to

- view and edit SAS data files
- customize the functionality of these classes for your own application development needs.

Unless you are customizing the table editor or you are using a model other than the original Data Set Data Model class provided, do not use this class. Use the Data Table class instead.

Before you customize the functionality of these classes, you should understand the structure of these classes and how they communicate.

The Table Editor class provides the ability to display, print, and interact with rectangular (tabular) data. The display and printed output are graphical, as proportional fonts can be used, and the interaction includes column and row resizing, and cell selection and editing.

The table editor can utilize data from a variety of sources. The data source is a separate object referred to as the model, which should be a subclass of SASHELP.FSP.TABLE_M.CLASS (for example, Data Set Data Model). The table editor is associated with a particular model using the attach method (for example, Data Set Data Model). This separation of data and view is known as the Model-View-Controller (MVC) paradigm.

The Table Editor compartmentalizes the data into four categories - column labels, row labels, titles, and data cells.

The column and row labels and the titles are collectively known as the label cells. Each of these label areas can contain a hierarchy. This is known as a multidimensional table. Each strip or level in the hierarchy is a separate dimension.

A dimension that does not span the entire table is known as a group. The column, row, and title dimensions are also referred to as the nonscrollable area of the table.
because they are always displayed as the table is scrolled. See the _setAttributes method for information on hiding these areas.

Scrolling only affects the data or scrollable area. The scroll thumb position always reflects the current position in the data, and its size reflects the percentage of the total number of elements (rows or columns) that are visible. You can add to the nonscrollable area by holding columns and rows. For more information see the _setHeldColumns and _setHeldRows methods.

The column and row dimensions may be unbounded or dynamic in the case where the total number of elements in the dimension is unknown or changing. Such tables are horizontally and/or vertically dynamic. For example, a data table object with an active WHERE clause can be called vertically dynamic. Only single-dimension hierarchies can be dynamic.

While the label dimensions can be hierarchical, the data area is simply a checkerboard of cells formed by the intersection of the leaf (bottom dimension) rows and columns. This area can be visually divided along these rows and columns by spanning horizontal and vertical lines, which are collectively known as the grid. You can control the color, style, and width of the grid lines via the methods prefixed by _setGrid.

Do not confuse grid lines with cell borders. A cell border is a line drawn around the boundary of a cell. The exterior edge of this border is flush with the cell’s boundary; thus, it does not intrude on any neighboring cells. Like grid lines, the color, style, and width of these lines can be controlled; see the methods that contain _border in their names. Also see the LABEL BUTTONS and DATA BUTTONS attributes to create borders with a push-button appearance. Note that grid lines are drawn before cell borders so that the borders will be on top of the grid.

In addition to borders, you can control cell colors, fonts, margins, and justification. The table editor allows you to define defaults for each of these attributes for both label and data cells. Methods prefixed with _setLabel affect label cells while data cell defaults are specified with the _setData methods. For example, you can use the _setDataBackgroundColor method to change the default background color for all data cells. Note that the model can override any of these defaults.

You can select cells interactively or programmatically. Single clicking on a cell establishes it as the current or active cell. The active cell is the cell to which keystrokes are directed. The table editor distinguishes the active cell with a highlighted border known as the active cell indicator, which can be moved with the cursor keys. You can also select or highlight a contiguous range of cells. Dragging the mouse highlights any touched cells establishing a selected area. Selecting a row or column highlights all of the cells contained within it. Clicking in any cell unselects the area.

Double clicking on a cell not only makes it active, but it enables cell edit mode if the cell is not protected. Cell edit mode provides within-cell editing – the text cursor is placed on the letter on which you clicked, and the left and right cursor keys move the text cursor rather than the active cell indicator. You can terminate cell edit mode by clicking on another cell, tabbing or moving the cursor to another cell, pressing return, issuing a command, or scrolling. You can also edit unprotected cells by making them active and typing a character. However in this case, the typed character replaces the previous contents of the cell, and the left and right cursor keys move you from cell to cell rather than from letter to letter. This can be useful when entering a row or column of data. Note that not all models allow editing.

You can resize rows and columns by grabbing on or near their respective borders. The mouse shape changes as it is moved across a border to indicate that it can be grabbed. This change in mouse shape is a result of cursor tracking being enabled for the table editor and thus allows you to also resize rows and/or columns. If you were to disable the ability to resize the rows and/or columns in your table editor, then cursor tracking would be turned off by default. Even label cells without displayed borders can
be resized; simply place the mouse cursor along the cell’s edge. Note that not all models allow resizing.

The term table editor is used as a general term and should be interpreted as a data table, a table editor, or a subclass of a table editor.

PARENT: SASHELP.FSP.WIDGET.CLASS
CLASS: sashelp.fsp.Table_e.class

Using the Table Editor Class

**Making a Table Editor**

By default, the table editor is not on the selection list displayed by the Make action on the pop-up menu in the frame. Therefore, to create a table editor object, do one of the following:

- From a command line, issue the command:
  
  `rm make 'table editor'

- Add the Table Editor class to the selection list of choices displayed by the Make action in the frame pop-up menu. Then you can create the object by selecting Table Editor from the list.

- If you are using your own RESOURCE entry, edit it and add the Table Editor class. Make sure the DISPLAY status is on. Table Editor will be added to the selection list.

- Otherwise, copy the default RESOURCE entry, SASHELP.FSP.BUILD RESOURCE, into your current catalog. Edit the RESOURCE entry, and make sure the DISPLAY status is on. Table Editor will be added to the selection list.

**Making Selections in Table Editor**

The table editor provides features that enable users to select one or more rows or columns in the table. Selection features determine whether multiple selected rows can be noncontiguous, called multiple selections, or contiguous, called an extended selection.

You enable selection features by setting attributes for the table. In addition to determining whether multiple rows or columns can be selected, other attributes determine whether a row can be selected by clicking any of its columns or a row or column can be selected by clicking its label.

**Multiple Selections**

Multiple selections refers to the ability to select or highlight more than one noncontiguous range of cells. The ability to make multiple selections can be enabled with the table attribute MULTIPLE SELECTIONS. By default, this attribute is not enabled.

Users can make multiple selections by pressing the CTRL key while selecting a cell or range of cells with a single mouse click. All selections can be cleared by either a single click anywhere in the table or pressing the SHIFT key while clicking the mouse once anywhere in the table.
By default, the current selected group of cells are registered as a drag site. However, the drag operation is disabled when there is more than one selection.

**Extended Selection**

Extended selection refers to the ability to select or highlight a range of contiguous cells, extending a selection from the active cell (starting anchor point) to the cell last clicked with the mouse (ending anchor point).

Users can make an extended selection by pressing the SHIFT key while selecting a cell with a single mouse click. To clear all selections, single click anywhere in the table.

Extended selection is useful when the anchor points of your selected range are far apart. Instead of dragging out the entire selection you can select the cell that is the starting anchor point with a single click, scroll the table, then press the SHIFT key and hold it down while you select the cell that is the ending anchor point with a single click.

**Row Selection Mode**

Row selection mode expands a selection of any cell, except column labels, into a selection of the entire row that contains the selected cell. Row selection mode can be enabled with the table attribute SELECT_ROWS. By default, this attribute is not enabled.

When both the attributes SELECT_ROWS and MULTIPLE_SELECTIONS are set, table users can select multiple noncontiguous rows by pressing the CTRL key while selecting a row with a single mouse click. Users can deselect a row by pressing the CTRL key and clicking on the selected row.

Row selection mode is useful for mimicking list box behavior.

**Column Selection Mode**

Column selection mode expands a selection of any cell, except row labels, into a selection of the entire column that contains the selected cell. Column selection mode can be enabled with the table attribute SELECT_COLUMNS. By default, this attribute is not enabled.

When both the attributes SELECT_COLUMNS and MULTIPLE_SELECTIONS are set, table users can select multiple noncontiguous columns by pressing the CTRL key while selecting a column with a single mouse click. A column can be deselected by pressing the CTRL key and clicking on the selected column.

**Label Selection Mode**

Label selection mode expands the selection of any row or column label into a selection of the entire row or column, respectively. Label selection mode is enabled with the table attribute SELECT_LABELS. By default, this attribute is not enabled.

When both the attributes SELECT_LABELS and MULTIPLE_SELECTIONS are set, users can select multiple noncontiguous rows or columns in the table by pressing the CTRL key while selecting a row or column label, respectively, with a single mouse click. Users can deselect a column or row by pressing the CTRL key and clicking on the selected element.

The SELECT_LABELS attribute also has three companion table attributes, SELECT_rowLABELS, SELECT_columnLABELS, and SELECT_TITLES. The SELECT_LABELS attribute treats label selections as selections of the corresponding row or column. Alternatively, you can enable this behavior individually for row label cells with SELECT_rowLABELS, column label cells with SELECT_columnLABELS, or title cells with SELECT_TITLES. Thus, the SELECT_LABELS table attribute is the
equivalent of enabling all three of these companion table attributes. By default, none of these companion table attributes are enabled.

Coordinate Lists

All of the methods that operate on a particular row, column, or cell have coordinate arguments. These arguments are SCL lists rather than simple numerics in order to deal with multi-dimensional data, that is, nested rows or columns. Each element in one of these lists contains an integer value. Thus, the lists are simply integer arrays where each integer is a row-column coordinate for the table dimension that corresponds to that element’s index number.

Units of Measure

Several of the methods accept a unit of measure argument. The following table lists the valid units and provides a definition:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>inches</td>
</tr>
<tr>
<td>cm</td>
<td>centimeters</td>
</tr>
<tr>
<td>mm</td>
<td>millimeters</td>
</tr>
<tr>
<td>pt</td>
<td>points</td>
</tr>
<tr>
<td>pc</td>
<td>picas</td>
</tr>
<tr>
<td>el</td>
<td>l-space (one-third the width of an em)</td>
</tr>
<tr>
<td>em</td>
<td>the width of a piece of type about as wide as it is tall; usually 1 ln in the x-direction</td>
</tr>
<tr>
<td>en</td>
<td>n-space (half the width of an em)</td>
</tr>
<tr>
<td>ex</td>
<td>x-space (height of the x-character)</td>
</tr>
<tr>
<td>fg</td>
<td>figure width (width of the zero-character)</td>
</tr>
<tr>
<td>sp</td>
<td>space (width of the space-character)</td>
</tr>
<tr>
<td>cc</td>
<td>width of the widest character in the font</td>
</tr>
<tr>
<td>ht</td>
<td>height of the tallest character in the font</td>
</tr>
<tr>
<td>dp</td>
<td>depth of the deepest character in the font</td>
</tr>
<tr>
<td>ln</td>
<td>line space (1ht+1dp)</td>
</tr>
</tbody>
</table>

Creating a Multi-dimensional Model Class for Use with the Table Editor

The Table Editor has the ability to display data from a model in a simple table or in a complex or multi-dimensional table. A model representing a rectangular data source such as a SAS dataset (for example, Data Set Data Model) will produce a simple table
while a model representing a hierarchical data source (for example, Multi-dimensional Data Base Model) can yield a complex or multi-dimensional table. In either case, it is up to the data model to convey the structure of its data source as well as the data itself to the Table Editor.

Communication between the Table Editor and its Model

Any class that provides the methods defined by Table Data Model can be used as a model for the Table Editor. The Table Data Model defines the methods that are required to communicate between the model and the Table Editor. The following are the methods that are called by the Table Editor on the model. These methods can be divided into two sections: request methods, which provide the Table Editor with information about the model's data source and notification methods, which provide the model with information about the Table Editor.

The request methods can be further divided into two categories. The first category of method provides information about data structure of the model's data source using the _getColumnDimInfo, _getRowDimInfo, and _getTitleDimInfo methods. The second category of methods provides the actual data content from the model's data source using the _getColumnInfo, _getRowInfo, _getTitleInfo, and _getData methods.

The Table Editor calls the methods that provide the data structure when the model is first attached to the Table Editor and then afterwards only if requested by the model. When the model's data structure changes, the model can inform the Table Editor by calling one of the following methods: _updateColumnDim, _updateRowDim, _update_titleDim and _updateDim. This will cause the Table Editor to re-query the data structure from the model.

The Table Editor calls the methods that provide the model's data content whenever the current view changes. Circumstances that cause the view to change include scrolling and resizing the table, font and margin changes, and explicit update requests from the model by calling one of the following methods of the Table Editor: _update, _updateColumnData, _updateColumnLabels, _updateData, _updateLabels, _updateRange, _updateRowData, _updateRowLabels, _updateTitleLabels.

Structure of the Columns and Rows

This example will show you how to create a multi-dimensional model that can be used with the table editor. Note, when using this model, you cannot move rows and columns within the Table Editor. The model created will be a subclass of Table Data Model class and assumes that the data used will be available in an SCL List form containing a hierarchy of lists. This list will be stored as an instance variable on the model named 'DATABASE', which the multi-dimensional model will use to retrieve the information about the row and columns data structure and contents.

In this example, the data being modeled has 3 top-level columns labeled 'CANADA', 'GERMANY' and 'USA'. In the second-level of the hierarchy, each of the top-level columns has 2 second-level columns labeled 'CONSUMER' and 'EDUCATION'. In the third-level of the hierarchy, each of the second-level columns, 'CONSUMER' and 'EDUCATION', has a third-level column labeled 'ACTUAL'.

There are 2 top-level rows labeled '1998' and '1999'. Each top-level row has 4 second-levels row labeled 'Quarter 1' through 'Quarter 4'. Within each second-level row, there is a numeric data value for each third-level column.

In the example below, the sublist 'COLUMNS' and the rows are stored in order within a sublist 'ROWS'.

```plaintext
DATABASE (COLUMNS=(CANADA=(CONSUMER=('ACTUAL' ) ) [L]
          EDUCATION=('ACTUAL'
```
GERMANY=(CONSUMER=('ACTUAL')
   EDUCATION=('ACTUAL'))
USA=(CONSUMER=('ACTUAL')
   EDUCATION=('ACTUAL'))
ROWS=(1998=(QUARTER 1=(16197
          15103
          15114
          15076
          14515
          13758
          )
       QUARTER 2=(14547
                   20027
                   14114
                   14898
                   13741
                   16032
                   )
       QUARTER 3=(14870
                   14983
                   15452
                   13052
                   13800
                   16892
                   )
       QUARTER 4=(15222
                   15252
                   13181
                   15200
                   16793
                   15954
                   )
       )
1999=(QUARTER 1=(13238
          15766
          15383
          14925
          15020
          14938
          )
       QUARTER 2=(12928
                   17887
                   16401
                   18479
                   15978
                   )
Warning: SAS guidelines prohibit nesting this deep Creating the DATABASE SCL List

The following is the code needed to create the SCL list DATABASE that will be used in this example. This code can be used one of two ways:

1. Include the code into the INIT label of the FRAME entry containing the Table Editor. This means that the SCL List DATABASE will be created each time the FRAME is executed.

2. Include the code into a separate SCL entry to create the list and then save the list as an SLIST entry. This SLIST entry can then be read within the INIT label of the FRAME entry containing the Table Editor.

Create the root list.

init:
rootlst = makelist();

Put the root columns into their own list to differentiate them from the rows.

columns = makelist();
rc = setniteml(rootlst, columns,
   'COLUMNS' );

   /* Create CANADA sublist for the */
   /* first column */
   canada = makelist();
   rc = setniteml(columns, canada,
      'CANADA' );
consumer = makelist();
rc = setniteml(canada, consumer,
   'CONSUMER' );
rc = setitemc(consumer, 'ACTUAL',
              1, 'y');
education = makelist();
rc = setniteml(canada, education,
              'EDUCATION');
rc = setitemc(education, 'ACTUAL',
              1, 'y');

/* Create GERMANY sublist for the */
/* second column */
germany = makelist();
rc = setniteml(columns, germany,
              'GERMANY');
consumer = makelist();
rc = setniteml(germany, consumer,
              'CONSUMER');
rc = setitemc(consumer, 'ACTUAL',
              1, 'y');
education = makelist();
rc = setniteml(germany, education,
              'EDUCATION');
rc = setitemc(education, 'ACTUAL',
              1, 'y');

/* Create USA sublist for the */
/* third column */
usa = makelist();
rc = setniteml(columns, usa,'USA');
consumer = makelist();
rc = setniteml(usa, consumer,
              'CONSUMER');
rc = setitemc(consumer, 'ACTUAL',
              1, 'y');
education = makelist();
rc = setniteml(usa, education,
              'EDUCATION');
rc = setitemc(education, 'ACTUAL',
              1, 'y');

Put the root rows into their own list to differentiate them from the columns.

rows = makelist();
rc = setniteml(rootlst, rows, 'ROWS');

Create the first row dimension for year 1999 and then create each of the quarter sublist along with values for that quarter:

year = makelist();
rc = setniteml(rows, year, '1998');
Create a Multi-dimensional Model Class for Use with the Table Editor

Creating a Multi-dimensional Model Class for Use with the Table Editor

Chapter 101

qtr = makelist();
rc = setniteml(year, qtr, 'Quarter 1');
rc = setitemn(qtr, 16197, 1, 'y');
rc = setitemn(qtr, 15103, 2, 'y');
rc = setitemn(qtr, 15114, 3, 'y');
rc = setitemn(qtr, 15076, 4, 'y');
rc = setitemn(qtr, 14515, 5, 'y');
rc = setitemn(qtr, 13758, 6, 'y');
qtr = makelist();
rc = setniteml(year, qtr, 'Quarter 2');
rc = setitemn(qtr, 14547, 1, 'y');
rc = setitemn(qtr, 20027, 2, 'y');
rc = setitemn(qtr, 14114, 3, 'y');
rc = setitemn(qtr, 14898, 4, 'y');
rc = setitemn(qtr, 13741, 5, 'y');
rc = setitemn(qtr, 16032, 6, 'y');
qtr = makelist();
rc = setniteml(year, qtr, 'Quarter 3');
rc = setitemn(qtr, 14870, 1, 'y');
rc = setitemn(qtr, 14983, 2, 'y');
rc = setitemn(qtr, 15452, 3, 'y');
rc = setitemn(qtr, 13052, 4, 'y');
rc = setitemn(qtr, 13800, 5, 'y');
rc = setitemn(qtr, 16892, 6, 'y');
qtr = makelist();
rc = setniteml(year, qtr, 'Quarter 4');
rc = setitemn(qtr, 15222, 1, 'y');
rc = setitemn(qtr, 15252, 2, 'y');
rc = setitemn(qtr, 13181, 3, 'y');
rc = setitemn(qtr, 15200, 4, 'y');
rc = setitemn(qtr, 16793, 5, 'y');
rc = setitemn(qtr, 15954, 6, 'y');

Create the second row dimension for year 1998 and then create each of the quarter sublist along with values for that quarter.

year = makelist();
rc = setniteml(rows, year, '1999');
qtr = makelist();
rc = setniteml(year, qtr, 'Quarter 1');
rc = setitemn(qtr, 13238, 1, 'y');
rc = setitemn(qtr, 15766, 2, 'y');
rc = setitemn(qtr, 15383, 3, 'y');
rc = setitemn(qtr, 14925, 4, 'y');
rc = setitemn(qtr, 15020, 5, 'y');
rc = setitemn(qtr, 14938, 6, 'y');
qtr = makelist();
rc = setniteml(year, qtr, 'Quarter 2');
rc = setitemn(qtr, 12928, 1, 'y');
rc = setitemn(qtr, 17887, 2, 'y');
rc = setitemn(qtr, 16401, 3, 'y');
rc = setitemn(qtr, 18479, 4, 'y');
rc = setitemn(qtr, 15978, 5, 'y');
crc = setitemn(qtr, 14130, 6, 'y');
qtr = makelist();
rc = setniteml(year, qtr, 'Quarter 3');
crc = setitemn(qtr, 15255, 1, 'y');
crc = setitemn(qtr, 15791, 2, 'y');
crc = setitemn(qtr, 16069, 3, 'y');
crc = setitemn(qtr, 16564, 4, 'y');
crc = setitemn(qtr, 14851, 5, 'y');
crc = setitemn(qtr, 14904, 6, 'y');
qtr = makelist();
rc = setniteml(year, qtr, 'Quarter 4');
crc = setitemn(qtr, 14804, 1, 'y');
crc = setitemn(qtr, 15351, 2, 'y');
crc = setitemn(qtr, 16229, 3, 'y');
crc = setitemn(qtr, 13354, 4, 'y');
crc = setitemn(qtr, 17601, 5, 'y');
crc = setitemn(qtr, 13631, 6, 'y');

If this is a standalone SCL entry used to create the list once then this line is needed to save the list out to the appropriate location.

rc = savelist( 'CATALOG',
              'MYLIB.MYCAT.DATABASE.SLIST',
              rootlst );

If this is a standalone SCL entry, delete the list that was created earlier. Otherwise, the list would be deleted in the TERM label of the FRAME.

rc = dellist( rootlst, 'Y' );
return;

Warning: SAS guidelines prohibit nesting this deep Creating the Multi-Dimensional Model Subclass to Read the DATABASE list

Now that the DATABASE list has been created, you need to create the multi-dimensional model subclass that will read the DATABASE list. To create a subclass of the Table Data Model and use this new class with a FRAME that contains a Table Editor object, follow these steps:

1 Subclass the Table Data Model class.
   a Edit multidim.class.
   b Set the description to 'My Multidimensional Class' and the parent class to 'SASHELP.FSP.TABLE_M_CLASS'.
   c Select 'Methods' under 'Additional Attributes' and override the following methods:
      □ _getColumnDimInfo method using the label of GETCDIM.
      □ _getColumnInfo method using the label of GETCOL.
      □ _getRowDimInfo method using the label of GETRDIM.
      □ _getRowInfo method using the label of GETROW.
_getData method using the label of GETDATA.

d Edit the SCL entry NEWTBLM.SCL using the source from 'Overriding Table Model methods for a multi-dim model' below.
e Compile and save the SCL entry.

2 Create a Table Editor object in a FRAME.

a Edit new.frame.
b Create an empty region in the FRAME entry.
c Use the command RM FILL 'TABLE EDITOR'.
d In the Table Editor Attribute Window set the name to 'Table' and OK from the window.
e Edit the FRAME’s SCL and use the source from 'Multidimensional Frame SCL'.
f Compile and save the FRAME.
g TESTAF the FRAME.

Warning: SAS guidelines prohibit nesting this deep

Overriding Table Data Model Methods for a Multi-dimensional Model

The following code overrides the methods from the Table Data Model for a multi-dimensional model.

Remove compile warnings.

```sas
length _method_ $ 50;
_self_ = _self_;```

Override the _getColumnDimInfo method of the Table Data Model.

```sas
/* This is a method override for the _getColumnDimInfo method
   of the Table Data Model, which provides the table editor with
   the column dimension hierarchy information. Parameters passed
   to this method:
   * 'level' - specifies the address of the column dimension being
     requested.
   * 'num_cols' - returns the number of columns that are present
     at the requested dimension.
   * 'sub_cols' - returns whether the requested dimension has any
     subdimensions.
   * 'height' - not used in this example.
   * 'units' - not used in this example.
   * 'groups' - not used in this example.
   * 'eod' - not used in this example. */
GETCDIM: method level num_cols sub_cols height 8 units $ 4 groups 8 eod $ 1;
/* Determine which column level is being requested and indicate
 * (1) How many columns (num_cols) the table has for this dimension.
 * (2) Whether this dimension has any subdimensions (sub_cols).
 * At the top level dimension, there are 3 columns and each has
 * subdimensions. */
len = listlen( level );
if ( '^len ) then
do;
```
num_cols = 3;
sub_cols = 1;
end;

/* At the second level dimension, there are 2 columns and each has
  * subdimensions. */
else if ( len = 1 ) then
do;
  num_cols = 2;
  sub_cols = 1;
end;

/* At a third level dimension, there is 1 column and with no
  * subdimensions. */
else
do;
  num_cols = 1;
  sub_cols = 0;
end;
endmethod;

Override the _getRowDimInfo method of the Table Data Model.

/* This is a method override for the _getRowDimInfo method
  * of the Table Data Model, which provides the table editor with
  * the row dimension hierarchy information. Parameters passed
  * to this method:
  *   'level' - specifies the address of the row dimension being
  *             requested.
  *   'num_rows' - returns the number of rows that are present at
  *                the requested dimension.
  *   'sub_rows' - returns whether the requested dimension has any
  *                subdimensions.
  *   'height' - not used in this example.
  *   'units' - not used in this example.
  *   'groups' - not used in this example.
  *   'eod' - not used in this example. */
GETRDIM: method level num_rows sub_rows width 8 units $ 4 groups 8 eod $ 1;

/* Determine the row level being requested and indicate:
  * (1) How many rows (num_rows) the table has in this
  *      dimension.
  * (2) Whether this dimension has subdimensions (sub_rows) */
len = listlen (level);
/* At the top level dimension, there are 2 rows and each row
  * has subdimensions. */
if ( ^len ) then
do;
  num_rows = 2;
  sub_rows = 1;
end;
/* At the second-level dimension, there are 4 rows and with no
  * subdimensions. */
else
do;
Override for the _getColumnInfo method of the Table Data Model.

```plaintext
/* This is a method override for the _getColumnInfo method of the Table Data Model, which provides the table editor with information about the column labels. Parameters passed to this method:
   'rcdvecid' - Use instance of the Row/Column data vector.
   The information about the requested column is set on the data vector. */
GETCOL: method rcdvecid 8;
   /* Retrieve the value of the instance variable DATABASE which contains the information for this multi-dimensional example. */
   From the DATABASE list, retrieve the COLUMNS sublist which contains the information about the columns. Within the COLUMNS list, the columns are stored in order starting at the beginning of the list. For example, the first column, CANADA, in the COLUMNS list is the first column to display in the table editor. */
   database = getniteml(_self_, 'DATABASE');
   columnlst = getniteml(database, 'COLUMNS');

   /* Use the _getCoordinates method of the Row/Column Data Vector to retrieve which column is currently being requested by the table editor for display. */
   coords = makelist();
   call send(rcdvecid, '_getCoordinates', coords);

   /* Get the top-level column dimension from the COORDS list */
   colnum = getitemn(coords, 1);

   /* Retrieve the second-level column dimension if available on the COORDS list. */
   if (listlen(coords) > 1) then
      subcolnum = getitemn(coords, 2);
   else
      subcolnum = 0;

   /* Retrieve the third-level column dimension if available on the COORDS list. */
   if (listlen(coords) > 2) then
      subsubcolnum = getitemn(coords, 3);
   else
      subsubcolnum = 0;

   /* The COORDS list is no longer needed. Delete the COORDS list. */
   coords = dellist(coords);

   /* Determine the label for the column to be displayed. */
   /* If Table Editor has requested three dimensions... */
```

```text
num_rows = 4;
sub_rows = 0;
endmethod;
```
if ( subsubcolnum ) then
do;
    /* Using the top-level column dimension(COLNUM), retrieve the */
    /* top-level dimension list from COLUMNS. For example, if */
    /* COLNUM is 1, then the sublist CANADA would be retrieved */
    /* from the COLUMNS list. */
    columnlst = getiteml( columnlst, colnum );

    /* Using the second-level column dimension(SUBCOLNUM), retrieve the */
    /* second-level dimension list from top-level list. For */
    /* example, if SUBCOLNUM is 2 and the top-level list was */
    /* CANADA, then the sublist CONSUMER would be retrieved from */
    /* the CANADA list. */
    columnlst = getiteml( columnlst, subcolnum );

    /* Since the last requested level is the third-level dimension, */
    /* you need retrieve the column label by getting the */
    /* value of the SUBCOLNUM item from the top-level list. */
    label = getitemc( columnlst, subsubcolnum );
end;
    /* If Table Editor has requested two dimensions... */
else if ( subcolnum ) then
do;
    /* Using the top-level column dimension(COLNUM), retrieve the */
    /* top-level dimension list from COLUMNS. For example, if */
    /* COLNUM is 1, then the sublist CANADA would be retrieved */
    /* from the COLUMNS list. */
    columnlst = getiteml( columnlst, colnum );

    /* Since the last requested level is the second-level dimension, */
    /* you need retrieve the column label by getting the */
    /* name of the SUBCOLNUM item from the top-level list. */
    label = nameitem( columnlst, subcolnum );
end;
    /* If Table Editor has requested one dimension... */
else
    /* Since the last requested level is the top-level dimension, */
    /* you need retrieve the column label by getting the */
    /* name of the COLNUM item from the COLUMNS list. */
    label = nameitem( columnlst, colnum );

    /* Set the label on the Row/Column Data Vector for the column. */
call send( rcdvecid, '_setText', label );
endmethod;

Override for the _getRowInfo method of the Table Data Model.

/* This is a method override for the _getRowInfo method */
/* of the Table Data Model, which provides the table editor with */
/* the information about the row labels. Parameters passed to this method: */
/* 'rcdvecid' - Use instance of the Row/Column data vector. */
/* The information about the requested row is */
/* set on the data vector. */
GETROW: method rcdvecid 8;
   /* Retrieve the value of the instance variable DATABASE
   * which contains the information for this multi-dimensional example.
   * From the DATABASE list, retrieve the ROWS sublist which
   * contains the information about the rows. Within the ROWS
   * list, the rows are stored in order starting at the beginning
   * of the list. For example, the first row, 1998, in the
   * ROWS list is the first row to display in the table editor. */
   database = getniteml( _self_, 'DATABASE' );
   rowlst = getniteml( database, 'ROWS' );
   
   /* Use the _getCoordinates method of the Row/Column Data Vector
   * to retrieve which row is currently being requested by the
   * table editor for display. */
   coords = makelist();
   call send ( rcdvecid, '_getCoordinates', coords );
   
   /* Get the first row dimension off of the COORDS list */
   rownum = getitemn( coords, 1 );
   
   /* Retrieve the other row subdimension if available on the
   * COORDS list. */
   if ( listlen( coords )=1 ) then
      subrownum = 0;
   else
      subrownum = getitemn( coords, 2 );
   
   /* The COORDS list is no longer needed. Delete the COORDS list. */
   coords = dellist( coords );
   
   /* Determine the label for the row to be displayed. */
   /* If Table Editor has requested two dimensions... */
   if ( subrownum ) then do;
      /* Using the top-level row dimension(ROWNUM), retrieve the
      * top-level dimension list from ROWS. For example, if
      * ROWNUM is 1, then the sublist 1998 would be retrieved
      * from the ROWS list. */
      rowlst = getiteml( rowlst, rownum );
      
      /* Since the last requested level is the second-level dimension,
      * you need retrieve the row label by getting the
      * name of the SUBROWNUM item from the top-level list. */
      label = nameitem( rowlst, subrownum );
      end;
   /* If Table Editor has requested one dimension... */
   else
      /* Since the last requested level is the top-level dimension,
      * you need retrieve the row label by getting the
      * name of the ROWNUM item from the ROWS list. */
      label = nameitem( rowlst, rownum );
   
   /* Set the label on the Row/Column Data Vector for the row. */
   call send( rcdvecid, '_setText', label );
Override for the _getData method of the Table Data Model.

/* This is a method override for the _getData method of the Table Data Model, which provides the table editor with contents of the data to be displayed. Parameters passed to this method:
   'gddvecid' - Use instance of the GET_DATA data vector class.
   The data vector is filled in with the contents of the requested row.
   data vector for 'num_cols'.
   'num_cols' - The number of columns to retrieve from the requested row. */

GETDATA: method gddvecid num_cols 8;

   /* Retrieve the value of the instance variable DATABASE which contains the information for this multi-dimensional example. From the DATABASE list, retrieve the ROWS sublist which contains the information about the rows. Within the ROWS list, the rows are stored in order starting at the beginning of the list. For example, the first row, 1998, in the ROWS list is the first row to display in the table editor. */
   database = getniteml( _self_, 'DATABASE' );
   rowlst = getniteml( database, 'ROWS' );

   /* Use the _getRow method of the GET_DATA Data Vector to retrieve which row is currently being requested by the table editor for display. */
   coords = makelist();
   call send( gddvecid, '_getRow', coords );

   /* Get the top-level row dimension from the COORDS list */
   rownum = getitemn( coords, 1 );

   /* Get the second-level row dimension from the COORDS list */
   subrownum = getitemn( coords, 2 );

   /* The COORDS list is no longer needed. Delete the COORDS list. */
   coords = dellist( coords );

   /* Using the top-level row dimension(ROWNUM), retrieve the top-level dimension list from ROWS. Then using the second-level row dimension(SUBBROWN), get the second-level list from the top-level list. For example, if ROWNUM is 1, then the sublist 1998 would be retrieved from the ROWS list. If SUBBROWN is 3, then the sublist 'Quarter 3' would be retrieved. */
   rowlst = getiteml( rowlst, rownum );
   rowlst = getiteml( rowlst, subrownum );

   /* From the second-level list retrieve the value for each of the passed columns. For each requested column: */
   (1) Get the column coordinates
* (2) Get the column text
* (3) Set the text in gddvecid */
colcoords = makelist();
doi = 1 to num_cols;

    /* Each time through the loop, we need to set the index for the
    * the current column in gddvecid. Then get the column
    * coordinates for that column. */
call send( gddvecid, '_setIndex', i );
call send( gddvecid, '_getColumn', colcoords );

    /* Get the top-level and second-level column dimensions from the
    * COLCOORDS list. */
colnum = getitemn( colcoords, 1 );
subcolnum = getitemn( colcoords, 2 );

    /* Calculate where to retrieve the data */
colnum = colnum * 2 - mod( subcolnum, 2 );

    /* Retrieve the data and convert it to a character string. */
text = trim( left( put( getitemn( rowlst, colnum ), 12. ) ) );

    /* Set the data in gddvecid */
call send( gddvecid, '_setText', text );
end;

    /* The COLCOORDS list is no longer needed. Delete the COLCOORDS list. */
colcoords = dellist( colcoords );
endmethod;

Multi-dimensional Frame SCL

The frame's SCL for the Multi-dimensional example is shown here.

Load your multi-dimensional class and create an instance of your multi-dimensional model.

rc = rc;
init:
    classid = loadclass
        ("MYLIB.MYCAT.MULTIDIM.CLASS");
call send( classid, "_new", modelid );

Read in the DATABASE list used for this example and add the DATABASE list as an instance variable on the model.

database = makelist();
rc = fillist( 'catalog',
    'MYLIB.MYCAT.DATABASE.SLIST',
database );
rc = setniteml( modelid, database,
    "DATABASE" );
Attach the instance of your multi-dimensional model to the table editor.

```javascript
    call notify( 'table', '_attach', modelid );
    return;
```

Detach the model from the Table Editor and terminate the multi-dimensional model.

```javascript
    term:
        call notify( 'table', '_detach' );
        call send( modelid, '_term' );
```

Delete the DATABASE list.

```javascript
    database = dellist( database );
    return;
```

Methods

Methods specific to the Table Editor class are described here. Inherited methods are described in the Widget class.

Dictionary

__attach__

Associates the specified data model with the table editor

Syntax

```javascript
CALL SEND (table-id, '_attach', data-id);
```

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data-id</td>
<td>N</td>
<td>specifies the object identifier of the data model to attach</td>
</tr>
</tbody>
</table>

Details

The __attach__ method associates the specified data object with the table editor object. The data object is referred to as the model and is used as the data source for the table. This
object is expected to be a subclass of the Table Data Model class (for example, Data Set Data Model), which is the abstract class that defines the communication protocol between the table editor and its attached model.

If the Table Editor is already attached to another model, that model is detached, and then the table is repopulated with the new model’s data.

**See Also**

_-detach_

### `_clearActiveCell`

Unselects the current active cell

**Syntax**

```call send```
(table-id, '_clearActiveCell');

### `_clearSelect`

Unselects the current highlighted area

**Syntax**

```call send```
(table-id, '_clearSelect');

### `_createHTML`

Converts the displayed table format to a Web page format viewable in any table-enabled Web browser

**Syntax**

```call send```
(table-id, '_createHTML', html-file);
Table Editor Class

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table-id</td>
<td>N</td>
<td>specifies the object identifier of the data table</td>
</tr>
<tr>
<td>html-file</td>
<td>C</td>
<td>specifies the name of the HTML file created by the method that contains the table</td>
</tr>
</tbody>
</table>

**Details**

The `_createHTML` method recreates the content of a data table in an HTML-formatted Web page. Note that a table-enabled browser is required to view the output from `_createHTML`.

**Using the `_createHTML` method**

This example adds the HTML table generation feature to a push button on a frame that also contains a data table.

1. Create a new frame. (You can use the command `build work.a.a.frame`).
2. Add a data table named 'table' to the frame. Modify the data table's object attributes to specify a table to display. (For example, set the Table field to `SASHELP.RETAIL.`)
3. Add a push button named 'htmlbtn' to the frame. Modify the push button's object attributes to specify a label for the button. (For example, you can label the button "Make HTML").
4. Edit the frame's SCL and add a labeled section for the push button's action:

   ```sas
   htmlbtn:
   call notify('table', '_createHTML', 'test.html');
   return;
   ```

5. Select
   
   [Run](Run) ➤ [Compile](Compile)
   
   to compile the frame's SCL, then select
   
   [Run](Run) ➤ [TestAF](TestAF)
   
   to test the application.

6. When the application executes, you can click the push button to generate the HTML for the table you specified. The resulting output is stored in the file 'test.html' in the directory from which you invoked SAS software.

---

**_detach**

Disassociates the currently attached data model

**Syntax**

`CALL SEND (table-id, '_detach');`
Details
A table editor with no attached data model yields an empty display.
The _detach method has no effect on the model object itself; that is, the object is not deleted.

See Also
_attach

_getActiveCell

Returns the current cell

Syntax
CALL SEND (table-id, '_getActiveCell', row, col);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>N</td>
<td>specifies the identifier of an SCL list to contain the row coordinates. See “Coordinate Lists” on page 1817.</td>
</tr>
<tr>
<td>col</td>
<td>N</td>
<td>specifies the identifier of an SCL list to contain the column coordinates. See “Coordinate Lists” on page 1817.</td>
</tr>
</tbody>
</table>

See Also
_setActiveCell

_getAttributes

Returns the current attributes for the table

Syntax
CALL SEND (table-id, '_getAttributes', attr-list);
Table Editor Class

### _getDataBackgroundPattern

Returns the default background pattern for data cells

**Syntax**

```
CALL SEND (table-id, '_getDataBackgroundPattern', pattern-name);
```

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pattern-name</td>
<td>C</td>
<td>returns the color name</td>
</tr>
</tbody>
</table>

### _getColumnConformThreshold

Returns the column conform threshold

**Syntax**

```
CALL SEND (table-id, '_getColumnConformThreshold', threshold);
```

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>threshold</td>
<td>N</td>
<td>returns the column conform threshold in integer values in the range 0 to 99</td>
</tr>
</tbody>
</table>

**Details**

The column conform threshold is a value in the range 0 to 99. This value, determined by the ratio of the table’s width to the region’s width, is the percentage above which the CONFORM_COLUMNS attribute takes effect.

### _getDataBackgroundColor

Returns the default background color for data cells

**Syntax**

```
CALL SEND (table-id, '_getDataBackgroundColor', color-name);
```

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color-name</td>
<td>C</td>
<td>returns the color name</td>
</tr>
</tbody>
</table>

### _getDataBackgroundPattern

Returns the default background pattern for data cells

**Syntax**

```
CALL SEND (table-id, '_getDataBackgroundPattern', pattern-name);
```

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pattern-name</td>
<td>C</td>
<td>returns the color name</td>
</tr>
</tbody>
</table>

| attr-list | N | specifies the identifier of an SCL list to contain the current values of all the named attributes listed in Table 101.4 on page 1864 |

**See Also**

_setAttributes
**Syntax**

CALL NOTIFY (table-id, '_getDataBackgroundColor', pattern);

**Where ...**  **Type**  **Description**

<table>
<thead>
<tr>
<th>pattern</th>
<th>C</th>
<th>returns the data background pattern:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>'SOLID'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'75%'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'50%'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'25%'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'TRANSPARENT'</td>
</tr>
</tbody>
</table>

**See Also**

_setDataBackgroundColor

---

**_getDataBorderColor**

Returns the default border color for data cells

**Syntax**

CALL SEND (table-id, '_getDataBorderColor', border-name, color-name);

**Where ...**  **Type**  **Description**

<table>
<thead>
<tr>
<th>border-name</th>
<th>C</th>
<th>specifies the desired border section:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>'TOP'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'LEFT'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'RIGHT'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'BOTTOM'</td>
</tr>
</tbody>
</table>

| color-name  | C       | returns the color name |

---

**_getDataBorderStyle**

Returns the default border style for data cells
Syntax

**CALL SEND** (table-id, '_getDataBorderStyle', border-name, style);

Where ... Type Description

| border-name | C | specifies the desired border section: |
|             |   | 'TOP'                              |
|             |   | 'LEFT'                             |
|             |   | 'RIGHT'                            |
|             |   | 'BOTTOM'                           |

| style       | C | returns the border style:          |
|             |   | 'SOLID'                            |
|             |   | 'DOTTED'                           |
|             |   | 'DASHED'                           |

**_getDataBorderStyle**

Returns the default border width for data cells

---

Syntax

**CALL SEND** (table-id, '_getDataBorderWidth', border-name, width-value, width-unit);

Where ... Type Description

| border-name | C | specifies the desired border section: |
|             |   | 'TOP'                              |
|             |   | 'LEFT'                             |
|             |   | 'RIGHT'                            |
|             |   | 'BOTTOM'                           |

| width-value  | N | returns the border width in width-unit units |
| width-unit   | C | returns the unit of measure. See "Units of Measure" on page 1817 for a complete list. |

**_getDataColor**

Returns the default text color for data cells
**Syntax**

**CALL SEND** (table-id, '_getDataColor', color-name);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color-name</td>
<td>C</td>
<td>returns the color name</td>
</tr>
</tbody>
</table>

---

### _getDataFont

Returns the default font for data cells

---

**Syntax**

**CALL SEND** (table-id, '_getDataFont', font-list-id);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>font-list-id</td>
<td>N</td>
<td>specifies the identifier of an SCL list to contain the font attributes</td>
</tr>
</tbody>
</table>

**Details**

Do not manipulate the contents of the font list directly, but instead pass the entire list to any method that sets fonts, for example, _setLabelFont.

---

### _getDataHjust

Returns the default horizontal justification for data cells

---

**Syntax**

**CALL SEND** (table-id, '_getDataHjust', just);
Table Editor Class

getDataMargin

Returns the default margin for data cells

Syntax

CALL SEND (table-id, 'getDataMargin', margin-name, margin-value, margin-unit);

getDataLightSource

Returns the default light source for data cells

Syntax

CALL NOTIFY (table-id, 'getDataLightSource', light-source);

getDataMargin

Returns the default margin for data cells

Syntax

CALL SEND (table-id, 'getDataMargin', margin-name, margin-value, margin-unit);
### `getDataVjust`

**Returns the default vertical justification for data cells**

**Syntax**

```
CALL SEND (table-id, 'getDataVjust', just);
```

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>just</td>
<td>C</td>
<td>returns the justification type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>TOP</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>MIDDLE</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>BOTTOM</code></td>
</tr>
</tbody>
</table>

### `_getGridColor`

**Returns the grid line color**

**Syntax**

```
CALL SEND (table-id, '_getGridColor', color-name);
```
Table Editor Class

_getHeldColumns

Returns the currently held columns

Syntax

CALL SEND (table-id, '_'getHeldColumns';

Where ... Type Description

- color-name  C  returns the grid line color name

_getGridStyle

Returns the grid line style

Syntax

CALL SEND (table-id, '_getGridStyle', style);

Where ... Type Description

- style  C  returns the grid line style:
  - 'SOLID'
  - 'DOTTED'
  - 'DASHED'

_getGridWidth

Returns the grid line width

Syntax

CALL SEND (table-id, '_getGridWidth', width-value, width-unit);

Where ... Type Description

- width-value  N  returns the line width in width-unit values
- width-unit  C  returns the unit of measure. See "Units of Measure" on page 1817 for a complete list.
**Syntax**

**CALL SEND** (table-id, '_getHeldColumns', start-col, end-col);

Where ...  Type  Description
---  ---  ---------------
start-col  N  specifies the identifier of an SCL list to contain the coordinates of the first held column. See “Coordinate Lists” on page 1817.
end-col  N  specifies the identifier of an SCL list to contain the coordinates of the last held column. See “Coordinate Lists” on page 1817.

**Details**

The lists are empty if no columns are held.

---

**_getHeldRows**

Returns the currently held rows

**Syntax**

**CALL SEND** (table-id, '_getHeldRows', start-row, end-row);

Where ...  Type  Description
---  ---  ---------------
start-row  N  specifies the identifier of an SCL list to contain the coordinates of the first held row. See “Coordinate Lists” on page 1817.
end-row  N  specifies the identifier of an SCL list to contain the coordinates of the last held row. See “Coordinate Lists” on page 1817.

**Details**

The lists are empty if no rows are held.

---

**_getHscroll**

Returns the horizontal scroll unit

**Syntax**

**CALL SEND** (table-id, '_getHscroll', unit<, num-units>);
### Where ...

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit</td>
<td>returns the scrolling unit:</td>
</tr>
<tr>
<td></td>
<td>'PAGE'</td>
</tr>
<tr>
<td></td>
<td>'HALF'</td>
</tr>
<tr>
<td></td>
<td>'MAX'</td>
</tr>
<tr>
<td></td>
<td>'COLUMN'</td>
</tr>
</tbody>
</table>

| num-units | returns the number of units to scroll |

### Details

If the unit is... the table scrolls horizontally...

- 'PAGE': by the number of visible columns
- 'HALF': by half the number of visible columns
- 'MAX': to the number of the maximum columns (see the _getMaxcol method in this class) minus the visible columns
- 'COLUMN': by the amount specified by num-units

### See Also

_setHscroll and _hscroll

---

### _getLabelBackgroundColor

Returns the default background color for label cells

#### Syntax

**CALL SEND** (table-id, 'getLabelBackgroundColor', color-name);

Where...

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color-name</td>
<td>returns the background color name</td>
</tr>
</tbody>
</table>

---

### _getLabelBackgroundPattern

Returns the default background pattern for label cells
**Syntax**

**CALL NOTIFY** (table-id, '_getLabelBackgroundColor', pattern);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pattern</td>
<td>C</td>
<td>returns the label background pattern:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'SOLID'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'75%'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'50%'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'25%'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'TRANSPARENT'</td>
</tr>
</tbody>
</table>

**See Also**

_setLabelBackgroundColor

---

**_getLabelBorderColor**

Returns the default border color for label cells

**Syntax**

**CALL SEND** (table-id, '_getLabelBorderColor', border-name, color-name);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>border-name</td>
<td>C</td>
<td>specifies the desired border section:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'TOP'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'LEFT'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'RIGHT'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'BOTTOM'</td>
</tr>
<tr>
<td>color-name</td>
<td>C</td>
<td>returns the color name</td>
</tr>
</tbody>
</table>

---

**_getLabelBorderStyle**

Returns the default border style for label cells
Syntax

CALL SEND (table-id, '_getLabelBorderStyle', border-name, style);

Where ... Type Description

border-name   C   specifies the desired border section:
                TOP
                'LEFT'
                'RIGHT'
                'BOTTOM'

style         C   returns the border style:
                'SOLID'
                'DOTTED'
                'DASHED'

_getLabelBorderWidth

Returns the default border width for label cells

Syntax

CALL SEND (table-id, '_getLabelBorderWidth', border-name, width-value, width-unit);

Where ... Type Description

border-name   C   specifies the desired border section:
                TOP
                'LEFT'
                'RIGHT'
                'BOTTOM'

width-value   N   returns the border width in width-unit values

width-unit    C   returns the unit of measure. See "Units of Measure" on page 1817 for a complete list.

_getLabelColor

Returns the default text color for label cells
Syntax

CALL SEND (table-id, '_getLabelColor', color-name);

Where ... Type Description
---------- ------ ---------------------------------
color-name C returns the color name

_getLabelFont

Returns the default font for label cells

Syntax

CALL SEND (table-id, '_getLabelFont', font-list-id);

Where ... Type Description
---------- ------ --------------------------------------------------------------------------------
font-list-id N specifies the identifier of an SCL list to contain a copy of a font list

Details

Do not manipulate the contents of the font list directly, but instead pass the entire list to any method that sets fonts, for example, _setLabelFont.

_getLabelHjust

Returns the default horizontal justification for label cells

Syntax

CALL SEND (table-id, '_getLabelHjust', just);
**_getLabelMargin**

Returns the default margin for label cells

**Syntax**

```
CALL SEND (table-id, '_getLabelMargin', margin-name, margin-value, margin-unit);
```

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>margin-name</td>
<td>C</td>
<td>returning the default margin value for label cells</td>
</tr>
<tr>
<td>margin-value</td>
<td>C</td>
<td>returning the default margin value for label cells</td>
</tr>
<tr>
<td>margin-unit</td>
<td>C</td>
<td>returning the default margin unit for label cells</td>
</tr>
</tbody>
</table>

**_getLabelLightSource**

Returns the default light source for label cells

**Syntax**

```
CALL NOTIFY (table-id, '_getLabelLightSource', light-source);
```

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>light-source</td>
<td>C</td>
<td>returning the default light source for label cells</td>
</tr>
<tr>
<td>light-source</td>
<td>C</td>
<td>returning the default light source for label cells</td>
</tr>
<tr>
<td>light-source</td>
<td>C</td>
<td>returning the default light source for label cells</td>
</tr>
<tr>
<td>light-source</td>
<td>C</td>
<td>returning the default light source for label cells</td>
</tr>
</tbody>
</table>

**_getLabelMargin**

Returns the default margin for label cells

**Syntax**

```
CALL SEND (table-id, '_getLabelMargin', margin-name, margin-value, margin-unit);
```

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>margin-name</td>
<td>C</td>
<td>returning the default margin value for label cells</td>
</tr>
<tr>
<td>margin-value</td>
<td>C</td>
<td>returning the default margin value for label cells</td>
</tr>
<tr>
<td>margin-unit</td>
<td>C</td>
<td>returning the default margin unit for label cells</td>
</tr>
</tbody>
</table>

**_getLabelLightSource**

Returns the default light source for label cells

**Syntax**

```
CALL NOTIFY (table-id, '_getLabelLightSource', light-source);
```

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>light-source</td>
<td>C</td>
<td>returning the default light source for label cells</td>
</tr>
<tr>
<td>light-source</td>
<td>C</td>
<td>returning the default light source for label cells</td>
</tr>
<tr>
<td>light-source</td>
<td>C</td>
<td>returning the default light source for label cells</td>
</tr>
<tr>
<td>light-source</td>
<td>C</td>
<td>returning the default light source for label cells</td>
</tr>
</tbody>
</table>

**_getLabelMargin**

Returns the default margin for label cells

**Syntax**

```
CALL SEND (table-id, '_getLabelMargin', margin-name, margin-value, margin-unit);
```

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>margin-name</td>
<td>C</td>
<td>returning the default margin value for label cells</td>
</tr>
<tr>
<td>margin-value</td>
<td>C</td>
<td>returning the default margin value for label cells</td>
</tr>
<tr>
<td>margin-unit</td>
<td>C</td>
<td>returning the default margin unit for label cells</td>
</tr>
</tbody>
</table>
Where ... Type Description
margin-name C specifies the desired margin:
   'TOP'
   'LEFT'
   'RIGHT'
   'BOTTOM'
margin-value N returns the size of the margin in margin-unit values
margin-unit C returns the unit of measure. See "Units of Measure" on page 1817 for a complete list.

_getLabelVjust

Returns the default vertical justification for label cells

Syntax
CALL SEND (table-id, '_getLabelVjust', just);

Where ... Type Description
just C returns the justification type:
   'TOP'
   'MIDDLE'
   'BOTTOM'

_getLeftcolumn

Returns the address of the left-most scrollable column

Syntax
CALL SEND (table-id, '_getLeftcolumn', col);
<table>
<thead>
<tr>
<th>Where</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
<td>N</td>
<td>specifies the identifier of an SCL list to contain the coordinates of the left-most scrollable column. See “Coordinate Lists” on page 1817.</td>
</tr>
</tbody>
</table>

### _getMaxcol

**Returns the maximum number of columns in the table**

#### Syntax

```
CALL SEND (table-id, '_getMaxcol', max);
```

<table>
<thead>
<tr>
<th>Where</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max</td>
<td>N</td>
<td>returns the maximum number of columns in the table. This number is negative if the table is horizontally dynamic.</td>
</tr>
</tbody>
</table>

#### Details

The _getMaxcol method returns the maximum number of columns for which a table can be scrolled right. A negative value indicates that the number of columns is dynamic. The ratio of this value to the number of visible columns (see the _getViscol method in this class) determines the size of the thumb in the horizontal scroll bar.

### _getMaxrow

**Returns the maximum number of rows in the table**

#### Syntax

```
CALL SEND (table-id, '_getMaxrow', max);
```

<table>
<thead>
<tr>
<th>Where</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max</td>
<td>N</td>
<td>returns the maximum number of rows in the table. This number is negative if the table is vertically dynamic.</td>
</tr>
</tbody>
</table>

#### Details

The _getMaxrow method returns the maximum number of rows for which a table can be scrolled down. A negative value indicates that the number of rows is dynamic. The
ratio of this value to the number of visible rows (see the _getVisrow method in this
class) determines the size of the thumb in the vertical scroll bar.

__getPopupCell__

Returns the row and column coordinates of the cell where the last pop-up occurred

**Syntax**

```
CALL SEND (table-id, '_getPopupCell', row, col);
```

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>N</td>
<td>specifies the identifier of an SCL list to contain the row coordinates. See “Coordinate Lists” on page 1817.</td>
</tr>
<tr>
<td>col</td>
<td>N</td>
<td>specifies the identifier of an SCL list to contain the column coordinates. See “Coordinate Lists” on page 1817.</td>
</tr>
</tbody>
</table>

**Details**

The pop-up cell is the most recent cell to have received a pop-up menu event. You
should call this method when overriding the _popup method in order to determine
which cell is being acted upon. If this method is called before any cell has received a
pop-up event, then empty lists are returned.

__getProperties__

Returns a list of information about the current state of both the table and the model

**Syntax**

```
CALL SEND (table-id, '_getProperties', prop-list);
```

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prop-list</td>
<td>N</td>
<td>specifies the identifier of an SCL list to contain the properties of the table</td>
</tr>
</tbody>
</table>

**Details**

The _getProperties method is useful for copying or re-creating a table. You can pass
this properties list to the _new method of a new table or the _setProperties method of
an existing table. The properties list also includes information specific to the data model. Note that some properties, such as held columns, are stored on the properties list with the NOWRITE attribute which prevents them from being written by the SCL SAVELIST function.

### _getRowConformThreshold

**Returns the row conform threshold**

**Syntax**

```
CALL SEND (table-id, '_getRowConformThreshold', threshold);
```

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>threshold</td>
<td>N</td>
<td>returns the row conform threshold in integer values in the range 0 to 99</td>
</tr>
</tbody>
</table>

**Details**

The row conform threshold is a value in the range 0 to 99. This value, determined by the ratio of the table's height to the region's height, is the percentage above which the CONFORM_ROWS attribute takes effect.

### _getSelect

**Returns the coordinates of the highlighted (selected) area**

**Syntax**

```
CALL SEND (table-id, '_getSelect', ULrow, ULcol, LRrow, LRcol);
```

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULrow</td>
<td>N</td>
<td>specifies the identifier of an SCL list to contain the upper-left row coordinates. See “Coordinate Lists” on page 1817.</td>
</tr>
<tr>
<td>ULcol</td>
<td>N</td>
<td>specifies the identifier of an SCL list to contain the upper-left column coordinates. See “Coordinate Lists” on page 1817.</td>
</tr>
<tr>
<td>Where ...</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>LRrow</td>
<td>N</td>
<td>specifies the identifier of an SCL list to contain the lower-right row coordinates. See &quot;Coordinate Lists&quot; on page 1817.</td>
</tr>
<tr>
<td>LRcol</td>
<td>N</td>
<td>specifies the identifier of an SCL list to contain the lower-right column coordinates. See &quot;Coordinate Lists&quot; on page 1817.</td>
</tr>
</tbody>
</table>

**Details**

Empty lists are returned if there is no selected area.

---

**_getSelections**

Returns a list of the selections.

---

**Syntax**

```
CALL SEND (object-id, '_getSelections', sel-list);
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sel-list</td>
<td>N</td>
<td>specifies the identifier of an SCL list to contain the selections.</td>
</tr>
</tbody>
</table>

**Details**

The selections returned by the _getSelections method are grouped into individual sublists, one for each selection. Each of the sublists within the outer list contains four named lists, START_ROW, START_COLUMN, END_ROW, and END_COLUMN. These named lists contain the coordinates for the particular point.

---

**Example**

The following example shows the contents of the SCL list returned by the _getSelections method when two noncontiguous ranges of cells in a table editor are selected or highlighted. The list shown below represents the following ranges:

range 1: row 1, column 1 through row 4, column 2

range 2: row 5, column 3 through row 8, column 4

```
SELECTIONS(1=(START_ROW=(1=1 ))[2029])
START_COLUMN=(1=1 ))[2027])
END_ROW=(1=4 )][2023])
END_COLUMN=(1=2 )][2097])
```
_getTableHjust

Returns the horizontal justification for the table

Syntax
CALL SEND (table-id, '_getTableHjust', just);

Where ... Type Description
just C returns the justification type:
'LEFT'
'CENTER'
'RIGHT'

_getTableVjust

Returns the vertical justification for the table

Syntax
CALL SEND (table-id, '_getTableVjust', just);
**_getToprow_**

Returns the address of the top-most scrollable row

**Syntax**

CALL SEND (table-id, '_getToprow', row);

Where ... | Type | Description
--- | --- | ---
row | N | specifies the identifier of an SCL list to contain the coordinates of the top-most scrollable row. See “Coordinate Lists” on page 1817.

**_getViscol_**

Returns the number of columns that are currently visible

**Syntax**

CALL SEND (table-id, '_getViscol', numcols<, partial-cols>);

Where ... | Type | Description
--- | --- | ---
numcols | N | returns the number of visible columns
partial-cols | N | returns the number of partially visible columns. The only possible values are 0 or 1.

**Details**

The value obtained by subtracting partial-cols from numcols is used when a table is scrolled horizontally by a PAGE or HALF. See the _setHscroll method in this class for more information. The ratio of this value to the maximum number of columns determines the size of the thumb in the horizontal scroll bar.
**_getVisrow_**

Returns the number of rows that are currently visible

**Syntax**

**CALL SEND** (table-id, '_getVisrow', numrows<, partial-rows>);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numrows</td>
<td>N</td>
<td>returns the number of visible rows</td>
</tr>
<tr>
<td>partial-rows</td>
<td>N</td>
<td>returns the number of partially visible rows. The only possible values are 0 or 1.</td>
</tr>
</tbody>
</table>

**Details**

The value obtained by subtracting partial-rows from numrows is used when a table is scrolled vertically by a PAGE or HALF. See the _setVscroll method in this class for more information. The ratio of this value to the maximum number of rows determines the size of the thumb in the vertical scroll bar.

**_getVscroll_**

Returns the vertical scroll unit

**Syntax**

**CALL SEND** (table-id, '_getVscroll', unit<, num-units>);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit</td>
<td>C</td>
<td>returns the scrolling unit: 'PAGE' 'HALF' 'MAX' 'ROW'</td>
</tr>
<tr>
<td>num-units</td>
<td>N</td>
<td>returns the number of units to scroll</td>
</tr>
</tbody>
</table>

**Details**
If the unit is... the table scrolls vertically...

'PAGE' by the number of visible rows
'HALF' by half the number of visible rows
'MAX' either to the top or the bottom
'ROW' by the amount specified by num-units

See Also
_setVscroll and _vscroll

_gotoCell

Goes to the specified cell

Syntax
CALL SEND(table-id, '_gotoCell', row, col <, upper-left>);

Where ...

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>specifies the identifier of an SCL list that contains the row coordinates. See “Coordinate Lists” on page 1817.</td>
</tr>
<tr>
<td>col</td>
<td>specifies the identifier of an SCL list that contains the column coordinates. See “Coordinate Lists” on page 1817.</td>
</tr>
<tr>
<td>upper-left</td>
<td>If specified and nonzero, then the cell is placed in the upper-left corner; otherwise, if the cell is already fully visible, then no scrolling occurs:</td>
</tr>
<tr>
<td>1</td>
<td>position the cell in the upper-left corner of the table</td>
</tr>
<tr>
<td>0</td>
<td>if the cell is already visible, don’t move it; otherwise, position it in the upper-left corner of the table</td>
</tr>
</tbody>
</table>

Details
The _gotoCell method ensures that a particular cell is visible. It does not make the cell active; see _setActiveCell in this class.

_gotoColumn

Goes to a specified column
Table Editor Class

Syntax

_CALL SEND (table-id, '_gotoColumn', col);

Where ... Type Description

col N specifies the identifier of an SCL list that contains the column coordinates. If the column is already fully visible, then no scrolling occurs; otherwise, the column is made the left-most scrollable column.

See Also

_setLeftcolumn

__gotoRow

Goes to the specified row

Syntax

_CALL SEND (table-id, '_gotoRow', row);

Where ... Type Description

row N specifies the identifier of an SCL list that contains the row coordinates. If the row is already fully visible, then no scrolling occurs; otherwise, the row is made the top-most scrollable row.

See Also

_setToprow

__hscroll

Scrolls the table horizontally

Syntax

_CALL SEND (table-id, '_hscroll', unit<, num-units>);
Where ... | Type | Description
--- | --- | ---
unit | C | specifies the scrolling unit:
    - 'PAGE'
    - 'HALF'
    - 'MAX'
    - 'COLUMN'
    - The default scroll unit is used if unit is not specified.
num-units | N | specifies the number of units to scroll. For scrolling right, specify a positive number. For scrolling left, specify a negative number. The default scroll amount is used if num-units is not specified.

Details

If the unit is... | the table scrolls horizontally...
--- | ---
'PAGE' | by the number of visible columns
'HALF' | by half the number of visible columns
'MAX' | to the value 1 or the number of the maximum columns (see the _getMaxcol method)
'COLUMN' | by the amount specified by num-units

See Also

_getHscroll and _setHscroll

_popup

Displays the run-mode pop-up menu

Syntax

CALL SEND (table-id, '_popup'fill-list, selection);

Argument | Type | Description
--- | --- | ---
fill-list | N | specifies the identifier of an SCL list that contains the pop-up menu items that will be appended to the default pop-up menu
selection | N | returns the selected pop-up menu item

Details

The _popup method for the table editor displays a run-mode pop-up menu. You can pass an empty list if you only want the default items in the pop-up menu. See the _popup method in the Widget class for more information and an example.
_print

Prints the table

---------

Syntax

CALL SEND(table-id, '_print', <device<, options>>);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>device</td>
<td>C</td>
<td>specifies the print device. You may specify a blank value for the device. Some hosts allow you to specify a default printer for your SAS session. If you do not specify a blank for the device and one is not set up, you will be queried for the device name each time a page is printed.</td>
</tr>
<tr>
<td>options</td>
<td>N</td>
<td>specifies the identifier of an SCL list that contains any or all of the named items listed in Table 101.1 on page 1859. Items that are not specified in the options list use the default value.</td>
</tr>
</tbody>
</table>

Table 101.1 _print Method Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'AREA'</td>
<td>specifies how much of the table to print:</td>
</tr>
<tr>
<td></td>
<td>'SCREEN'</td>
</tr>
<tr>
<td></td>
<td>'ALL'</td>
</tr>
<tr>
<td>'MAJOR'</td>
<td>specifies the order in which to print multiple pages:</td>
</tr>
<tr>
<td></td>
<td>'ROW'</td>
</tr>
<tr>
<td></td>
<td>'COLUMN'</td>
</tr>
</tbody>
</table>

This is only used if 'AREA' is set to 'ALL'.

Details

The table is printed graphically and independently of the FSFORMs subsystem. The table is printed with the best representation possible of the way the table actually appears on the screen.

If no arguments are specified, the table is printed with the options specified in the Print window or with the _printDialog method.
See Also

_printSetup and _printDialog

_printDialog

Brings up the Print window to initialize printing options and print

Syntax

CALL SEND (obj1, '_printDialog', options);

Where ...

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>N</td>
</tr>
</tbody>
</table>

Table 101.2 _printDialog Method Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'AREA'</td>
<td>specifies how much of the table to print: 'SCREEN' print the current screen only (default) 'ALL' print all pages</td>
</tr>
<tr>
<td>'MAJOR'</td>
<td>specifies the order in which to print multiple pages: 'ROW' print complete rows before printing the next row (default) 'COLUMN' print complete columns before printing the next column This is only used if 'AREA' is set to 'ALL'.</td>
</tr>
<tr>
<td>'DEVICEN'</td>
<td>specifies the print device. You may specify a blank value for the device. Some hosts allow you to specify a default printer for your SAS session. If you do not specify a blank for the device and one is not set up, you will be queried for the device name each time a page is printed.</td>
</tr>
</tbody>
</table>

Details

Note that the _printDialog method can initiate printing. It also saves options for later use by the _print method or the Print pop-up menu item.
_printPreview

Displays the formatted table as it will appear when printed.

Syntax

CALL SEND (table-id, '_printPreview'<, options>);

Where ... Type Description

| options | N    | specifies the identifier of an SCL list that contains any or all of the named items listed in Table 101.2 on page 1860. On input, the items in options initialize the Print window. On output, the list reflects the selections made in the Print window. Items that are not specified in the options list use the default value. |

Table 101.3 _printDialog Method Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'AREA'</td>
<td>specifies how much of the table to print: 'SCREEN' print the current screen only (default) 'ALL' print all pages</td>
</tr>
<tr>
<td>'MAJOR'</td>
<td>specifies the order in which to print multiple pages: 'ROW' print complete rows before printing the next row (default) 'COLUMN' print complete columns before printing the next column This is only used if 'AREA' is set to 'ALL'.</td>
</tr>
</tbody>
</table>

_printSetup

Displays the host specific Print Setup window.

Syntax

CALL SEND (table-id, '_printSetup');
See Also

_print and _printDialog

_releasColumn

Releases currently held columns and returns to normal scrolling

Syntax

CALL SEND (table-id, '_releaseColumn');

See Also

_setHeldColumns

_releaseRow

Releases currently held rows and returns to normal scrolling

Syntax

CALL SEND (table-id, '_releaseRow');

See Also

_setHeldRows

_selectAll

Highlights the entire table

Syntax

CALL SEND (table-id, '_selectAll');

__selectColumn

Highlights the given column range
Syntax

**CALL SEND** (table-id, `_selectColumn`, start-col<, end-col>);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start-col</td>
<td>N</td>
<td>specifies the identifier of an SCL list that contains the coordinates of the first column to highlight. See “Coordinate Lists” on page 1817.</td>
</tr>
<tr>
<td>end-col</td>
<td>N</td>
<td>specifies the identifier of an SCL list that contains the coordinates of the last column to highlight. See “Coordinate Lists” on page 1817.</td>
</tr>
</tbody>
</table>

**_selectRow**

Highlights the given row range

Syntax

**CALL SEND** (table-id, `_selectRow`, start-row<, end-row>);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start-row</td>
<td>N</td>
<td>specifies the identifier of an SCL list that contains the coordinates of the first row to highlight. See “Coordinate Lists” on page 1817.</td>
</tr>
<tr>
<td>end-row</td>
<td>N</td>
<td>specifies the identifier of an SCL list that contains the coordinates of the last row to highlight. See “Coordinate Lists” on page 1817.</td>
</tr>
</tbody>
</table>

**_setActiveCell**

Sets the current cell

Syntax

**CALL SEND**(table-id, `_setActiveCell`, row, col);
_setAttributes

Sets the attributes of the table

Syntax

CALL SEND (table-id, '_setAttributes', attr-list);

Where ... Type Description

attr-list N specifies the identifier of an SCL list that contains any number of
the named attributes listed in the table provided in Table 101.4 on
page 1864.

Table 101.4 Attribute List

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'COLUMN_LABELS'</td>
<td>C</td>
<td>'Y'</td>
<td>indicates the display of column label cells.</td>
</tr>
<tr>
<td>'CONFORM_COLUMNS'</td>
<td>C</td>
<td>'N'</td>
<td>indicates conformed columns. When conformed columns are enabled, if the width of the data table is less than the width of the region, then the columns are resized proportionally to include the extra space; thus, the data table is flush with the right edge of the region. See also _setColumnConformThreshold.</td>
</tr>
<tr>
<td>'CONFORM_ROWS'</td>
<td>C</td>
<td>'N'</td>
<td>indicates conformed rows. When conformed rows are enabled, if the height of the data table is less than the height of the region, then the rows are resized proportionally to include the extra space; thus, the data table is flush with the bottom edge of the region. See also _setRowConformThreshold.</td>
</tr>
<tr>
<td>'DATA_BUTTONS'</td>
<td>C</td>
<td>'N'</td>
<td>indicates the display of data cells with a push-button-like border.</td>
</tr>
<tr>
<td>Attribute Name</td>
<td>Type</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>'DOUBLE_CLICK'</td>
<td>C</td>
<td>'N'</td>
<td>indicates whether a double click event drives the data table's object label.</td>
</tr>
<tr>
<td>'EDIT'</td>
<td>C</td>
<td>'N'</td>
<td>indicates whether the data from the model can be edited.</td>
</tr>
<tr>
<td>'GRID'</td>
<td>C</td>
<td>'Y'</td>
<td>indicates the display of grid lines.</td>
</tr>
<tr>
<td>'IN_CELL_CONTROLS'</td>
<td>C</td>
<td>'Y'</td>
<td>indicates the display of host controls (such as a spin box or combo box)</td>
</tr>
<tr>
<td>'LABEL_BUTTONS'</td>
<td>C</td>
<td>'Y'</td>
<td>indicates the display of label cells with a pushbutton-like border.</td>
</tr>
<tr>
<td>'MULTIPLE_SELECTIONS'</td>
<td>C</td>
<td>'N'</td>
<td>indicates whether multiple selections are allowed.</td>
</tr>
<tr>
<td>'PARTIAL_COLUMNS'</td>
<td>C</td>
<td>'Y'</td>
<td>indicates the display of partial columns.</td>
</tr>
<tr>
<td>'PARTIAL_ROWS'</td>
<td>C</td>
<td>'Y'</td>
<td>indicates the display of partial rows.</td>
</tr>
<tr>
<td>'RESIZE_columnLABELS'</td>
<td>C</td>
<td>'N'</td>
<td>indicates the ability to resize column label cell heights interactively.</td>
</tr>
<tr>
<td>'RESIZE_COLUMNS'</td>
<td>C</td>
<td>'N'</td>
<td>indicates the ability to resize column widths interactively.</td>
</tr>
<tr>
<td>'RESIZE_rowLABELS'</td>
<td>C</td>
<td>'N'</td>
<td>indicates the ability to resize row label cell widths interactively.</td>
</tr>
<tr>
<td>'RESIZE_ROWS'</td>
<td>C</td>
<td>'N'</td>
<td>indicates the ability to resize row heights interactively.</td>
</tr>
<tr>
<td>'ROW_LABELS'</td>
<td>C</td>
<td>'Y'</td>
<td>indicates the display of row label cells.</td>
</tr>
<tr>
<td>'SELECT_activeCELL'</td>
<td>C</td>
<td>'Y'</td>
<td>indicates the ability to select an active cell.</td>
</tr>
<tr>
<td>'SELECT_columnLABELS'</td>
<td>C</td>
<td>'N'</td>
<td>indicates whether a selection of a column label will be expanded into a selection of the entire column.</td>
</tr>
<tr>
<td>'SELECT_COLUMNS'</td>
<td>C</td>
<td>'N'</td>
<td>indicates whether a selection of any cell, except row labels, will be expanded into a selection of the entire column containing the cell.</td>
</tr>
<tr>
<td>'SELECT_LABELS'</td>
<td>C</td>
<td>'N'</td>
<td>indicates the ability to select (highlight) rows and columns by single clicking on their corresponding label cell. Setting this attribute is equivalent to setting the attributes of SELECT_rowLABELS, SELECT_columnLABELS, and SELECT_TITLES to 'Y'.</td>
</tr>
</tbody>
</table>
### _setColumnConformThreshold

Sets the column conform threshold

**Syntax**

```
CALL SEND (table-id, '_setColumnConformThreshold', threshold);
```

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>threshold</td>
<td>N</td>
<td>specifies the column conform threshold in integer values in the range 0 to 99</td>
</tr>
</tbody>
</table>

**Details**

The column conform threshold is a value in the range 0 to 99. This value, determined by the ratio of the table’s width to the region’s width, is the percentage above which the CONFORM_COLUMNS attribute takes effect. The default is zero. For example, a 50% threshold causes the table to grow horizontally only if it is wider than half its region.

### _setDataBackgroundColor

Sets the default background color for data cells

**Syntax**

```
CALL SEND (table-id, '_setDataBackgroundColor', color-name);
```
Table Editor Class

setDataBorderColor

Specifies the border color for data cells.

Syntax

CALL NOTIFY (table-id, '_setDataBorderColor', color-name);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color-name</td>
<td>C</td>
<td>specifies the color name</td>
</tr>
</tbody>
</table>

Details

This method has no visual effect on button borders (see the DATA_BUTTONS and LABEL_BUTTONS attributes) because the border color is derived from the cell’s background color. Button borders have two light sides and two dark sides. The light sides are a lighter shade of the cell background color, while the dark sides are a darker shade. The like-colored sides always meet at a corner, for example, top left and bottom right. The position of the light sides corresponds with the direction of the light source, which can be controlled with the _setDataLightSource and _setLabelLightSource methods.

setDataBackgroundPattern

Specifies the background pattern for data cells.

Syntax

CALL NOTIFY (table-id, '_setDataBackgroundPattern', pattern);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pattern</td>
<td>C</td>
<td>specifies the data background pattern:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'SOLID'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'75%'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'50%'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'25%'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'TRANSPARENT'</td>
</tr>
</tbody>
</table>

Details

The percentages (25%, 50%, 75%) create a mix of the table's background and the region's background. For example, a yellow region background, a red table background, and 50% produce what appears to be a solid orange background. 100% is an alias for SOLID, and 0% is an alias for TRANSPARENT.

setDataBorderColor

Sets the default border color for data cells.
Syntax

**CALL SEND** (table-id, '_setDataBorderColor', border-name, color-name);

Where ... Type Description

border-name C specifies the desired border section:

- 'TOP'
- 'LEFT'
- 'RIGHT'
- 'BOTTOM'

color-name C specifies the color name

---

**setDataBorderStyle**

Sets the default border style for data cells

Syntax

**CALL SEND** (table-id, '_setDataBorderStyle', border-name, style);

Where ... Type Description

border-name C specifies the desired border section:

- 'TOP'
- 'LEFT'
- 'RIGHT'
- 'BOTTOM'

style C specifies the border style:

- 'SOLID'
- 'DOTTED'
- 'DASHED'

---

**setDataBorderWidth**

Sets the default border width for data cells
Syntax
CALL SEND (table-id, '_setDataBorderWidth', border-name, width-value, width-unit);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>border-name</td>
<td>C</td>
<td>specifies the desired border section:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'TOP'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'LEFT'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'RIGHT'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'BOTTOM'</td>
</tr>
<tr>
<td>width-value</td>
<td>N</td>
<td>specifies the border width in width-unit units</td>
</tr>
<tr>
<td>width-unit</td>
<td>C</td>
<td>specifies the unit of measure. See &quot;Units of Measure&quot; on page 1817 for a complete list.</td>
</tr>
</tbody>
</table>

Details
Font-dependent units of measure are resolved on a cell-by-cell basis in order to account for each cell’s font.

_setDataColor

Sets the default text color for data cells

Syntax
CALL SEND (table-id, '_setDataColor', color-name);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color-name</td>
<td>C</td>
<td>specifies the color name</td>
</tr>
</tbody>
</table>

_setDataFont

Sets the default font for data cells

Syntax
CALL SEND (table-id, '_setDataFont', font-list-id);
**Details**

A font list is acquired from the SCL FONTSEL function or a call to the `_getDataFont` method, the `_getLabelFont` method, or a similar method that gets fonts for other objects.

---

**_setDataHjust**

Sets the default horizontal justification for data cells

**Syntax**

`CALL SEND (table-id, '_setDataHjust', just);`

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>font-list-id</td>
<td>N</td>
<td>specifies the identifier of an SCL list that contains the font list</td>
</tr>
</tbody>
</table>

**Just**

C specifies the justification type:

- `LEFT`
- `CENTER`
- `RIGHT`

---

**_setDataLightSource**

Specifies the light source for data cells

**Syntax**

`CALL NOTIFY (table-id, '_setDataLightSource', light-source);`
Table Editor Class

### light-source

C

specifies the data light source:
- 'LOWER LEFT'
- 'LOWER RIGHT'
- 'UPPER LEFT'
- 'UPPER RIGHT'

### _setDataMargin

Sets the default margin for data cells

**Syntax**

```
CALL SEND (table-id, '_setDataMargin', margin-name, margin-value, margin-unit);
```

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>margin-name</td>
<td>C</td>
<td>specifies the desired margin:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'TOP'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'LEFT'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'RIGHT'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'BOTTOM'</td>
</tr>
<tr>
<td>margin-value</td>
<td>N</td>
<td>specifies the size of the margin in margin-unit values</td>
</tr>
<tr>
<td>margin-unit</td>
<td>C</td>
<td>specifies the unit of measure. See &quot;Units of Measure&quot; on page 1817 for a complete list.</td>
</tr>
</tbody>
</table>

**Details**

Font-dependent units of measure are resolved on a cell-by-cell basis.

### _setDataVjust

Sets the default vertical justification for data cells

**Syntax**

```
CALL SEND (table-id, '_setDataVjust', just);
```

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### _setGridColor

Sets the grid line color

**Syntax**

```
CALL SEND (table-id, '_setGridColor', color-name);
```

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color-name</td>
<td>C</td>
<td>specifies the grid line color name</td>
</tr>
</tbody>
</table>

### _setGridStyle

Sets the grid line style

**Syntax**

```
CALL SEND (table-id, '_setGridStyle', style);
```

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| style      | C    | specifies the grid line style:
|            |      | 'SOLID'
|            |      | 'DOTTED'
|            |      | 'DASHED'

### _setGridWidth

Sets the grid line width
Syntax

CALL SEND (table-id, '_setGridWidth', width-value, width-unit);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>width-value</td>
<td>N</td>
<td>specifies the line width in width-unit values</td>
</tr>
<tr>
<td>width-unit</td>
<td>C</td>
<td>specifies the unit of measure. See “Units of Measure” on page 1817 for a complete list.</td>
</tr>
</tbody>
</table>

Details

Font-dependent units of measure are resolved using the default data cell font.

_setHeldColumns

Holds a given range of columns

Syntax

CALL SEND (table-id, '_setHeldColumns', start-col, end-col);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start-col</td>
<td>N</td>
<td>specifies the identifier of an SCL list that contains the coordinates of the first column to hold. See “Coordinate Lists” on page 1817.</td>
</tr>
<tr>
<td>end-col</td>
<td>N</td>
<td>specifies the identifier of an SCL list that contains the coordinates of the last column to hold. See “Coordinate Lists” on page 1817.</td>
</tr>
</tbody>
</table>

Details

The _setHeldColumns method makes a contiguous range of columns nonscrollable. The first column in this range is positioned flush against the row labels. These columns remain displayed just like row labels as you scroll horizontally. To resume normal scrolling, use the _releaseColumn method. The specified range replaces any previously held range.
**See Also**

_setHeldRows

### _setHeldRows

Holds a given range of rows

#### Syntax

```call send (table-id, '_setHeldRows', start-row, end-row);```

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start-row</td>
<td>N</td>
<td>specifies the identifier of an SCL list that contains the coordinates of the first row to hold. See “Coordinate Lists” on page 1817.</td>
</tr>
<tr>
<td>end-row</td>
<td>N</td>
<td>specifies the identifier of an SCL list that contains the coordinates of the last row to hold. See “Coordinate Lists” on page 1817.</td>
</tr>
</tbody>
</table>

#### Details

The _setHeldRows method makes a contiguous range of rows nonscrollable. The first row in this range is positioned flush against the column labels. These rows remain displayed just like column labels as you scroll vertically. To resume normal scrolling, use the _releaseRow method. The specified range replaces any previously held range.

**See Also**

_releaseRow

---

**_setHscroll**

Sets the default horizontal scrolling amount

#### Syntax

```call send (table-id, '_setHscroll', unit<, num-units>);```
### Table Editor Class

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| unit       | C    | specifies the scrolling unit:  
|            |      | 'PAGE'  
|            |      | 'HALF (default)'  
|            |      | 'MAX'  
|            |      | 'COLUMN'  |

| num-units  | N    | specifies the number of units to scroll. The default is 1. |

### Details

The scrolling unit and amount specified with the _setHscroll method are the defaults for the _hscroll method.

### See Also

_setHscroll and _hscroll

---

#### _setLabelBackgroundColor

**Sets the default background color for label cells**

---

**Syntax**

CALL SEND (table-id, 'setLabelBackgroundColor', color-name);

---

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color-name</td>
<td>C</td>
<td>specifies the background color name</td>
</tr>
</tbody>
</table>

### Details

This method has no visual effect on button borders (see the DATA_BUTTONS and LABEL_BUTTONS attributes) because the border color is derived from the cell’s background color. Button borders have two light sides and two dark sides. The light sides are a lighter shade of the cell background color, while the dark sides are a darker shade. The like-colored sides always meet at a corner, for example, top left and bottom right. The position of the light sides corresponds with the direction of the light source, which can be controlled with the _setDataLightSOURCE and _setLabelLightSource methods.

---

#### _setLabelBackgroundPattern

**Specifies the background pattern for label cells**
**Syntax**

**CALL NOTIFY** (table-id, 'setLabelBackgroundPattern', pattern);

Where ... Type Description

<table>
<thead>
<tr>
<th>pattern</th>
<th>C</th>
<th>specifies the label background pattern:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>'SOLID'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'75%'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'50%'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'25%'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'TRANSPARENT'</td>
</tr>
</tbody>
</table>

**Details**

The percentages (25%, 50%, 75%) create a mix of the table’s background and the region’s background. For example, a yellow region background, a red table background, and 50% produce what appears to be a solid orange background. 100% is an alias for SOLID, and 0% is an alias for TRANSPARENT.

---

**setLabelBorderColor**

Sets the default border color for label cells

**Syntax**

**CALL SEND** (table-id, '_setLabelBorderColor', border-name, color-name);

Where ... Type Description

<table>
<thead>
<tr>
<th>border-name</th>
<th>C</th>
<th>specifies the desired border section:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>'TOP'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'LEFT'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'RIGHT'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'BOTTOM'</td>
</tr>
</tbody>
</table>

| color-name | C | specifies the color name |

---

**setLabelBorderStyle**

Sets the default border style for label cells
Syntax

**CALL SEND** (table-id, 'setLabelBorderStyle', border-name, style);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>border-name</td>
<td>C</td>
<td>specifies the desired border section: ‘TOP’ ‘LEFT’ ‘RIGHT’ ‘BOTTOM’</td>
</tr>
<tr>
<td>style</td>
<td>C</td>
<td>specifies the border style: ‘SOLID’ ‘DOTTED’ ‘DASHED’</td>
</tr>
</tbody>
</table>

_setLabelBorderWidth

Sets the default border width for label cells

Syntax

**CALL SEND** (table-id, '_setLabelBorderWidth', border-name, width-value, width-unit);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>border-name</td>
<td>C</td>
<td>specifies the desired border section: ‘TOP’ ‘LEFT’ ‘RIGHT’ ‘BOTTOM’</td>
</tr>
<tr>
<td>width-value</td>
<td>N</td>
<td>specifies the border width in width-unit values</td>
</tr>
<tr>
<td>width-unit</td>
<td>C</td>
<td>specifies the unit of measure. See “Units of Measure” on page 1817 for a complete list.</td>
</tr>
</tbody>
</table>

Details

Font-dependent units of measure are resolved on a cell-by-cell basis in order to account for each cell’s font.
_setLabelColor

Sets the default text color for label cells

Syntax

CALL SEND (table-id, '_setLabelColor', color-name);

Where ... Type Description

color-name C specifies the color name

_setLabelFont

Sets the default font for label cells

Syntax

CALL SEND (table-id, '_setLabelFont', font-list-id);

Where ... Type Description

font-list-id N specifies the identifier of an SCL list that contains the font list

Details

A font list is acquired from the SCL FONTSEL function or a call to the _getDataFont method or the _getLabelFont method.

See Also

_setDataFont

_setLabelHjust

Sets the default horizontal justification for label cells

Syntax

CALL SEND (table-id, '_setLabelHjust', just);
### _setLabelLightSource_

Specifies the light source for label cells

---

**Syntax**

```call
CALL NOTIFY (table-id, '_setLabelLightSource', light-source);
```

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>light-source</td>
<td>C</td>
<td>specifies the label light source:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'LOWER LEFT’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'LOWER RIGHT’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'UPPER LEFT’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'UPPER RIGHT’</td>
</tr>
</tbody>
</table>

### _setLabelMargin_

Sets the default margin for label cells

---

**Syntax**

```call
CALL SEND (table-id, '_setLabelMargin', margin-name, margin-value, margin-unit);
```

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>just</td>
<td>C</td>
<td>specifies the justification type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'LEFT’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'CENTER’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'RIGHT’</td>
</tr>
<tr>
<td>Where ...</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>margin-name</td>
<td>C</td>
<td>specifies the desired margin: 'ALL’</td>
</tr>
<tr>
<td>margin-value</td>
<td>N</td>
<td>specifies the size of the margin in margin-unit values</td>
</tr>
<tr>
<td>margin-unit</td>
<td>C</td>
<td>specifies the unit of measure. See “Units of Measure” on page 1817 for a complete list.</td>
</tr>
</tbody>
</table>

**Details**

Font-dependent units of measure are resolved on a cell-by-cell basis.

---

**_setLabelVjust**

Sets the default vertical justification for label cells

**Syntax**

CALL SEND (table-id, '_setLabelVjust', just);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>just</td>
<td>C</td>
<td>specifies the justification type: 'TOP’</td>
</tr>
</tbody>
</table>

---

**_setLeftcolumn**

Sets the left-most scrollable column

**Syntax**

CALL SEND (table-id, '_setLeftcolumn', col);
Table Editor Class

_setMaxcol

Sets the maximum number of table columns

Syntax

CALL SEND (table-id, '_setMaxcol', max);

Where ... Type Description
max N specifies the maximum number of columns that can be displayed by scrolling right. A negative number makes the table horizontally dynamic.

Details

This method is not valid for multidimensional tables.

_setMaxrow

Sets the maximum number of table rows

Syntax

CALL SEND (table-id, '_setMaxrow', max);

Where ... Type Description
max N specifies the maximum number of rows that can be displayed by scrolling down. A negative number makes the table vertically dynamic.

Details

This method is not valid for multidimensional tables.
**_setMsg**

Specifies the text of the message

---

**Syntax**

CALL SEND (object-id, '_setMsg', msg_string);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msg_string</td>
<td>C</td>
<td>specifies the string of the message to be displayed.</td>
</tr>
</tbody>
</table>

**Details**

The message is displayed on the message line of the frame unless two or more messages have been issued since the last window refresh in which case the message will be sent to the log.

---

**_setProperties**

Restores the previous state of the table and the attached model

---

**Syntax**

CALL SEND (table-id, '_setProperties', prop-list);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prop-list</td>
<td>N</td>
<td>specifies the identifier of an SCL list that has been filled in by the _getProperties method</td>
</tr>
</tbody>
</table>

**Details**

The _setProperties method enables you to restore a previously saved table. It should only be called with a list provided by _getProperties. The model is notified of any model-specific properties contained in this list through its own _setProperties method.

---

**_setRowConformThreshold**

Sets the row conform threshold
Syntax

CALL SEND (table-id, '_setRowConformThreshold', threshold);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>threshold</td>
<td>N</td>
<td>specifies the column conform threshold in integer values in the range 0 to 99</td>
</tr>
</tbody>
</table>

Details

The row conform threshold is a value in the range 0 to 99. This value, determined by the ratio of the table’s height to the region’s height, is the percentage above which the CONFORM_ROWS attribute takes effect. The default is zero. For example, a 50% threshold causes the table to grow vertically only if it is taller than half its region.

_setSelect

Creates a highlighted area

Syntax

CALL SEND (table-id, '_setSelect', ULrow, ULcol, LRrow, LRcol);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULrow</td>
<td>N</td>
<td>specifies the identifier of an SCL list that contains the upper-left row coordinates. See “Coordinate Lists” on page 1817.</td>
</tr>
<tr>
<td>ULcol</td>
<td>N</td>
<td>specifies the identifier of an SCL list that contains the upper-left column coordinates. See “Coordinate Lists” on page 1817.</td>
</tr>
<tr>
<td>LRrow</td>
<td>N</td>
<td>specifies the identifier of an SCL list that contains the lower-right row coordinates. See “Coordinate Lists” on page 1817.</td>
</tr>
<tr>
<td>LRcol</td>
<td>N</td>
<td>specifies the identifier of an SCL list that contains the lower-right column coordinates. See “Coordinate Lists” on page 1817.</td>
</tr>
</tbody>
</table>

Details

Because the table editor maintains only one selected area at a time, each call to _setSelect clears any previously selected area. Use the _clearSelect method to clear the selection.
**_setTableHjust**

Sets justification type for horizontally justifying the table within the region

**Syntax**

CALL SEND (table-id, '_setTableHjust', just);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>just</td>
<td>C</td>
<td>specifies the justification type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'LEFT'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'CENTER'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'RIGHT'</td>
</tr>
</tbody>
</table>

**Details**

The _setTableHjust method enables you to control the horizontal position of the table within its region: flush left, centered, or flush right. It has no effect on tables that are wider than their region.

**_setTableVjust**

Sets justification type for vertically justifying the table within the region

**Syntax**

CALL SEND (table-id, '_setTableVjust', just);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>just</td>
<td>C</td>
<td>specifies the justification type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'TOP'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'MIDDLE'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'BOTTOM'</td>
</tr>
</tbody>
</table>

**Details**

The _setTableVjust method enables you to control the vertical position of the table within its region: flush top, centered, or flush bottom. It has no effect on tables that are taller than their region.
_setToprow

Sets the top-most scrollable row

Syntax
CALL SEND (table-id, '_setToprow', row);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| row        | N    | specifies the identifier of an SCL list that contains the row coordinates. See “Coordinate Lists” on page 1817.

_setVscroll

Sets the unit by which a table can scroll vertically

Syntax
CALL SEND (table-id, '_setVscroll', unit<, num-units>);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit</td>
<td>C</td>
<td>specifies the scrolling unit: ‘PAGE (default)’ ‘MAX’ ‘HALF’ ‘ROW’</td>
</tr>
</tbody>
</table>
| num-units      | N    | specifies the number of units to scroll. The default is 1.

Details
The scrolling unit and amount specified with the _setVscroll method are the defaults for the _vscroll method.

See Also
_getVscroll and _vscroll

_update

Queries the data model for label and data cells and refreshes the display
Syntax
CALL SEND (table-id, '_update');

_updateColumnData
Queries the data model for the given column range and refreshes the display

Syntax
CALL SEND (table-id, '_updateColumnData', start-col, end-col);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start-col</td>
<td>N</td>
<td>specifies the identifier of an SCL list that contains the coordinates of the first column to update. See &quot;Coordinate Lists&quot; on page 1817.</td>
</tr>
<tr>
<td>end-col</td>
<td>N</td>
<td>specifies the identifier of an SCL list that contains the coordinates of the last column to update. See &quot;Coordinate Lists&quot; on page 1817.</td>
</tr>
</tbody>
</table>

_updateColumnDim
Queries the data model for the column dimension hierarchy and refreshes the display

Syntax
CALL SEND (table-id, '_updateColumnDim');

_updateColumnLabels
Queries the data model for column label cells and refreshes the display

Syntax
CALL SEND (table-id, '_updateColumnLabels');
_updateData

Queries the data model for data cells and refreshes the display

Syntax
CALL SEND (table-id, '_updateData');

_updateDim

Queries the data model for the column, row, and title dimension hierarchies and refreshes the display

Syntax
CALL SEND (table-id, '_updateDim');

_updateLabels

Queries the data model for label cells and refreshes the display

Syntax
CALL SEND (table-id, '_updateLabels');

_updateRange

Queries the data model for the given cell range and refreshes the display
Syntax

CALL SEND (table-id, '_updateRange', ULrow, ULcol, LRrow, LRcol);

Where ... Type Description
ULrow   N  specifies the identifier of an SCL list that contains the upper-left row coordinates. See “Coordinate Lists” on page 1817.
ULcol   N  specifies the identifier of an SCL list that contains the upper-left column coordinates. See “Coordinate Lists” on page 1817.
LRrow   N  specifies the identifier of an SCL list that contains the lower-right row coordinates. See “Coordinate Lists” on page 1817.
LRcol   N  specifies the identifier of an SCL list that contains the lower-right column coordinates. See “Coordinate Lists” on page 1817.

_updateRowData

Queries the data model for the given row range and refreshes the display

Syntax

CALL SEND (table-id, '_updateRowData', start-row<, end-row>);

Where ... Type Description
start-row N  specifies the identifier of an SCL list that contains the coordinates of the first row to update
end-row   N  specifies the identifier of an SCL list that contains the coordinates of the last row to update

See Also

_updateColumnData

_updateRowDim

Queries the data model for the row dimension hierarchy and refreshes the display

Syntax

CALL SEND (table-id, '_updateRowDim');
_updateRowLabels
Queries the data model for row label cells and refreshes the display

Syntax
CALL SEND (table-id, '_updateRowLabels');

_updateTitleDim
Queries the data model for the title dimension hierarchy and refreshes the display

Syntax
CALL SEND (table-id, '_updateTitleDim');

_updateTitleLabels
Queries the data model for title label cells and refreshes the display

Syntax
CALL SEND (table-id, '_updateTitleLabels');

_vscroll
Scrolls the table vertically

Syntax
CALL SEND (table-id, '_vscroll', unit<, num-units>);
<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit</td>
<td>C</td>
<td>specifies the scrolling unit:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'PAGE'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'HALF'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'MAX'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'ROW'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The default scroll unit is used if unit is not specified.</td>
</tr>
<tr>
<td>num-units</td>
<td>N</td>
<td>specifies the number of units to scroll. For scrolling down, specify a positive number. For scrolling up, specify a negative number. The default scroll amount is used if num-units is not specified.</td>
</tr>
</tbody>
</table>

### Details

If the unit is... the table scrolls vertically...

- 'PAGE' by the number of visible rows
- 'HALF' by half the number of visible rows
- 'MAX' either to the top or the bottom (see the _getMaxrow method in this class)
- 'ROW' by the amount specified by num-units

### See Also

_getVscroll and_setVscroll