Overview

The Data Table class provides an interactive environment that enables you to browse and edit SAS tables. A data table provides a tabular display of multiple rows (observations) of a SAS table using the table you specify in the Table field of the Data Table Attributes window.

By adding a data table to your frame, you can add the capabilities of browsing and editing SAS tables without any writing complex SCL programs. However, you can use SCL to customize data form objects.

The Data Table class is a subclass of the Table Editor class. The only difference between the Data Table class and the Table Editor class is that the Data Table class automatically loads an instance of the Data Set Data Model class. Unlike the data table, the Table Editor class does not automatically load any model class unless you select one in the attributes window.

Parent:

sashelp.fsp.table_e.class

Class:

sashelp.fsp.DataTabl.class
Data Form and Data Table Concepts

Terminology

The Data Form and Data Table classes make use of several key terms and concepts. A SAS data set is a collection of data values organized into a table-like structure. The data values are organized into rows called observations and columns called variables. As needed for clarity, the terms table, row, and column are used to describe the Data Form and Data Table classes instead of the terms data set, observation, and variable.

Relative row number refers to the relative position of the row within the table (ignoring any deleted rows, and taking into account any WHERE clause or index key). Absolute row number refers to the actual physical location of the row within the table (taking into account deleted rows and ignoring and WHERE clause or index key). Absolute row number is not supported by all SAS data files.

Controlling Execution in the SCL

Program execution in the SCL entry assigned to a Data Set Data Model object is controlled by grouping statements into sections. Each section of the program begins with an SCL label and ends with a RETURN statement. Data Set Data Model object application programs can include the following six labeled sections:

1. **DFINIT**: an initialization phase before any rows are displayed, marked by the label DFINIT. This labeled section is run when the _setSource method of the Data Set Data Model class is called. This section is invoked only once for each SCL entry. Typical uses of this label are to:
   - import values through macro variables
   - display initial messages on the message line (using the _setMsg method)
   - open auxiliary data sets or external files used or referenced by the application.

   You cannot assign initial values to columns (variables displayed in the attached viewer, whether computed columns or data set columns) in the DFINIT section. Only columns which are not part of the data display can be initialized in this section because no rows are open during the processing of this block.

   The FSEINIT label is synonymous with the DFINIT label; if both labels are present, only the DFINIT label is run.

2. **INIT**: an initialization phase before each row is displayed, marked by the label INIT. This section is executed once per row, when the _getData method is called for the row. Typical uses of the statements following the INIT label are to:
   - initialize columns for the row
   - initialize computed columns.

   Note: If the viewer being used is a table editor, then the INIT label runs for each row that is displayed in the table. For example, if the table displays 10 rows, then INIT runs 10 times, executing the SCL code in it each time. △

3. **Column Labels**: a processing phase that runs when a given column is modified, marked by a label corresponding to a column name. Column labels for any
modified columns are run before the MAIN processing phase. Column labels allow you to separate specialized data validation code for each column in the table.

4 MAIN: a phase to process user input, marked by the label MAIN. This section is repeated each time the user modifies a field in the window or the _setData method of the object is driven by the attached viewer. When a field is modified, the Data Set Data Model class
  - checks the input values for validity. The Data Set Data Model class uses any column attributes assigned for the column.
  - executes the SCL statements following the MAIN label only if the input values are valid. If errors are detected, then statements are not executed and the class issues an error message. When the user corrects the error, the MAIN section is executed.

Note: Column values are checked for validity only before the SCL program is entered, not after. No error is detected if manipulations in the SCL program produce a field value that is outside the range specified in the MIN and MAX column attributes.

During the main processing phase, the user interacts directly with the application to accomplish specific tasks. The SCL program can prompt the user for information, verify values, check data sets, and call other programs that prompt the user for information.

5 TERM: a termination phase, marked by the label TERM, that executes before moving from a locked row. This label is driven when the _getUpdateStatus method of the Data Set Data Model class is called.
  
A typical use of the TERM section is to update an auxiliary data set.

6 DFTERM: a termination phase before the object terminates or when a new SCL entry is assigned to the object, marked by the label DFTERM. This section is executed when the _term method or _setSource method is called.
  
Typical uses of the statements following the DFTERM label are to
  - close tables or external files used or referenced by the application
  - export values through macro variables.

The FSETERM label is synonymous with the DFTERM label; if both labels are present, only the DFTERM label is run.

The dataform SCL respects the following SCL CONTROL statement arguments:
  - LABEL
  - ERROR
  - ALWAYS
  - ENTER
  - TERM

See SAS Component Language: Reference for more information on the CONTROL statement.

---

**Effect of Protecting Columns**

You can prevent users from modifying a column or columns by "protecting" the column or columns. Columns can be protected by using the _protectColumn method or by setting the Protected attribute for that column or columns. Note that the effect of protecting columns is different in a data form than in a data table. If you are using a data form, the effect of protecting a column is that you can type into columns that are
protected, but the widgets revert to their previous state when you press enter. In addition, in a data form you cannot tab to or tab out of a protected column. If you are using a data table, the effect of protecting a column is that you can not type into the column at all.

When a data form or data table is in browse mode, all of the columns are protected. In both the data form and data table, labeled sections for protected columns do not run.

How are "viewers" and "models" used in Data Form and Data Table?

The functionality of the Data Form and Data Table classes is accomplished by the use of a model and viewer. The model retrieves data and delivers it to the viewer for display. Generally, the model controls any action that is directly related to the data that is being displayed, for example, the column labels that are used in the form or table, the columns from the table (data set) that are included in the display, which rows for the columns are displayed, and the format associated with the data values. The viewer, on the other hand, controls the display and how users interact with it, for example, the movement within the form or table, including scrolling or paging through the columns.

Because the functionality is accomplished by using both a model and a viewer, the model and viewer must be able to communicate. The term used for this connection is attached, that is the model is attached to the viewer. When you use the Data Form and Data Table classes, the model is automatically attached for you. The model that is used with Data Form and Data Table is the Data Set Data Model class. Because the model is automatically attached, you can call both the model methods and the viewer methods on the data form or data table.

What is the model SCL?

The model SCL is separate from the frame's associated SCL entry. The model SCL entry is associated with a particular data form or data table object. It initializes computed columns and performs error checking and data validation. It also allows you to reference the columns by name.

Editing and Compiling the Model SCL

The model SCL for a data form or data table is associated with the data form or data table through the SCL Entry field in the object's attributes window. Specify the name of the model SCL entry in this field.

CAUTION:

Use unique names for the frame SCL entry and the model SCL entry. Do not specify the same SCL entry for both the frame SCL entry and the model SCL entry.

To edit the model SCL for a data form or data table, display the build-mode pop-up menu for the object, select

Form

(or)

Table

if the object is a data table), and then select...
Data Table Class

What is the model SCL?

The model SCL (Service Control Language) is a programming language used to create data forms and data tables in a specific system. It allows for the customization of the look and behavior of these forms and tables.

**Editing the SCL**

- Note that the 'Edit SCL...' item will be grayed out if you have not specified an SCL entry in the attributes window for the data form or data table.
- To compile the model SCL for a data form or data table, you can use one of the following methods:
  - Issue the COMPILE command from the EDIT SCL window.
  - Select `Form` or `Table` from the build-time pop-up menu and then select `Compile SCL`.
- **Note:** Compiling the model SCL from the BUILD window will cause an error at runtime.

**Using SCL Variables in Model SCL**

You can use the following three types of variables in your SCL program:

- **Column variables**
  - Variables that correspond to columns in the table, and computed columns, regardless of display status.
  - **Note:** You do not need to declare these in an SCL program.
  - Each column in the model has a corresponding column variable in SCL. When a user modifies the values in the columns of the object, the values of the corresponding SCL variables are automatically updated. Similarly, when the SCL program modifies the value of a window variable, the value of the corresponding column in the object is updated. If the column is for a data set column, the variable in the data set is updated also.

- **Noncolumn variables**
  - Variables that do not correspond to columns in the object. These include temporary variables used in the program, such as variables used to hold the return codes of SCL functions. These variables can be used in calculations and are treated by SCL in the same manner as column variables.
  - Noncolumn variables are initialized to missing values or to initial values given in declarative statements (such as ARRAY statements) before SCL is executed.

- **System variables**
  - Reserved variables such as `_SELF_` provided by SCL to check information set by the system or to set information displayed or used by the system. Four system variables can be used in the model SCL:
    - `_SELF_`, which contains the model’s object identifier. The model methods for Data Form and Data Table show the first argument as object-id. However, if you are calling these methods from the model SCL, you can substitute the system variables, `_SELF_`, for object-id. `_SELF_` can only be used with model methods.
When are data sets opened?

The data set associated with a data form or data table is opened at build-time using the mode and level defined in the object attributes window.

**DATAFORM Catalog Entries**

DATAFORM catalog entries can contain column customizations for the following classes: Data Table, Data Form, Table Editor, and Form Editor. In addition, DATAFORM catalog entries also contain layout information for all of these classes. (For more information on table editors and form editors, see the Table Editor Class and Form Editor Class.)

You can specify that a data table and a data form use the same DATAFORM catalog entry. This approach enables customizations to be shared between both classes with the use of a single catalog entry.

To specify a DATAFORM catalog entry, enter its name in the **DATAFORM Entry** field of the Data Table or Data Form Attributes window.

**CAUTION:**

Customizations can be lost. Whether customizations that you make are saved to a DATAFORM catalog entry is determined by what you specify in the **DATAFORM Entry** field in the Attributes window and by when you make the specification. DATAFORM
entries are saved each time you either exit the Attributes window, test the frame, or end from the frame.

For example, when you initially open the Attributes window, you can enter the name of a catalog entry in the DATAFORM Entry field or you can leave the field blank. Then, you can make customizations to your frame object. When you return to the Attributes window, you again have an opportunity to enter the name of a catalog entry in the DATAFORM Entry field.

For details on how changing the name of the catalog entry in the DATAFORM entry field can save or lose you customizations, see the following table.

Table 62.1  Customizations and DATAFORM Catalog Entries

<table>
<thead>
<tr>
<th>Before customizations, if the DATAFORM Entry field contains...</th>
<th>After you make customizations and specify a name for the DATAFORM entry as...</th>
<th>Then, the customizations are...</th>
</tr>
</thead>
<tbody>
<tr>
<td>nothing (blank)</td>
<td>a new DATAFORM catalog entry</td>
<td>saved to the new DATAFORM catalog entry. In a data form, the layout is not changed.</td>
</tr>
<tr>
<td>nothing (blank)</td>
<td>an existing DATAFORM catalog entry</td>
<td>lost; the object uses any customizations from the existing DATAFORM catalog entry. In a data form, the previous layout is lost and replaced by the saved layout in the new DATAFORM entry.</td>
</tr>
<tr>
<td>the name of a new DATAFORM catalog entry</td>
<td>another new DATAFORM catalog entry</td>
<td>saved to the new DATAFORM catalog entry that you specify after you make customizations. In a data form, the layout is not changed.</td>
</tr>
<tr>
<td>the name of a new DATAFORM catalog entry</td>
<td>an existing DATAFORM catalog entry</td>
<td>lost; the object uses any customizations from the existing DATAFORM catalog entry. In a data form, the previous layout is lost and replaced by the saved layout in the new DATAFORM entry.</td>
</tr>
<tr>
<td>the name of an existing DATAFORM catalog entry</td>
<td>a new DATAFORM catalog entry</td>
<td>lost; the object uses default customizations. In a data form, the previous layout is lost and a new default layout will be created for you.</td>
</tr>
<tr>
<td>the name of an existing DATAFORM catalog entry</td>
<td>another existing DATAFORM catalog entry</td>
<td>lost; the object uses any customizations from the existing DATAFORM catalog entry that you specify after you make customizations. In a data form, the previous layout is lost and replaced by the saved layout in the new DATAFORM entry.</td>
</tr>
</tbody>
</table>

Using the Data Table Class

Moving Columns in a Data Table

You can rearrange columns in a data table by following these steps:

1 Select the column or columns by clicking the select mouse button on the column's label and dragging the mouse until all of the columns you want to move are highlighted.
Drag the column or columns onto the column that you want them to follow and release the mouse button.

In some cases, the new location for your dragged column is not currently displayed; it is either to the left or to the right of the displayed columns, and you need to scroll to the new location. You can easily scroll to the new location at the same time that you are dragging a column by simply moving your cursor to the edge of the data table object as you drag the column. This will cause the data table object to "power scroll" to columns that aren't currently visible. When the new location is visible, drop the dragged column onto the column that you want it to follow and release the mouse button.

If you wish to change the display order of the columns in a Data Table and have that order saved then follow these steps:

1. Assign a Dataform entry to the Data Table so that customizations will be saved.
2. In BUILD mode, move the columns around to obtain the desired display order.

The column order you set in build mode will be used each time the Data Table is displayed.

You can also use SCL to do this at run time by using the _setDisplayedColumns method.

### Locking Rows in a Data Table

Many actions, such as editing a row, require that you first lock the row in the data table. By default, to lock a row in a data table, double-click the row.

### Making Selections in a Data Table

Data table users can select one or more rows or columns in the table when you have enabled these features. 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 users can make multiple selections, other attributes determine whether users can select a row by clicking on any of its columns or select a row or column by clicking on 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.

You can make multiple selections by pressing the CTRL key before making any selections and holding it down while selecting a cell or range of cells with a single mouse click. See the _getSelections method for information on retrieving the coordinates of the highlighted (selected) cells. To clear all selections, you can either single click anywhere in the table or press the SHIFT key and hold it down while you single click 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).

You can make an extended selection by first pressing the SHIFT key and then holding it down 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 a data table has both the attributes SELECT_ROWS and MULTIPLE_SELECTIONS, table users can select multiple noncontiguous rows by first pressing the CTRL key and holding it down 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 a data table has both the attributes SELECT_COLUMNS and MULTIPLE_SELECTIONS, table users can select multiple noncontiguous columns by pressing the CTRL key and holding it down while selecting a column with a single mouse click. Users can deselect a column 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 can be enabled with the table attribute SELECT_LABELS. By default, this attribute is not enabled.

When a data table has both the attributes SELECT_LABELS and MULTIPLE_SELECTIONS, users can select multiple noncontiguous rows or columns in the table by pressing the CTRL key and holding it down 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 table 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_titleLABELS. 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.

---

**Pop-up Menus for Data Table**

**Build-mode Pop-up Menu**

You can use your mouse to open the standard widget build pop-up menu with the Table item added. The build-mode pop-up menu items specific to the Data Table class are defined as follows:

- **Edit SCL**
  - opens a source window that contains the model’s associated SCL. This item is grayed when there is no associated SCL entry specified in the Data Table attributes window. To compile this SCL issue the COMPIL command from the EDIT SCL window.

- **Compile SCL**
  - compiles the associated SCL. The item is grayed when there is no associated SCL entry specified in the Data Table Attributes window.

- **Clear Active Cell**
  - deselects the active cell. This item is grayed when there is no active cell.

- **Clear Selections**
  - deselects the selected (highlighted) area. This item is grayed when there is no selected area.

- **Print Setup**
  - displays the host-specific Print Setup window.

- **Print**
  - displays the Print window that asks for the necessary information to print the table.

**Run-mode Pop-up Menu**

You can use your mouse to display a pop-up menu in run mode. To turn off the run-mode pop-up menu, select Command Processing from the Data Table Attributes window. Then, select Honor General Attribute from the Popmenu Processing list box.

The pop-up menu items for the Data Table class are defined as follows:

- **Clear Active Cell**
  - deselects the active cell. This item is grayed when there is no active cell.

- **Clear Selections**
  - deselects the selected (highlighted) area. This item is grayed when there is no selected area.

- **Print Setup**
  - displays the host-specific Print Setup window.
Print

displays the Print window that asks for the necessary information to print the table.

Methods

The Data Table methods include model methods and viewer methods. These are listed separately below.

Model Methods

Dictionary

_addComputedColumn

Adds a computed column that gets its value from SCL

Syntax

CALL SEND (object-id, '_addComputedColumn', attr-list);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr-list</td>
<td>N</td>
<td>specifies the identifier of an SCL list that contains the attributes for the computed column</td>
</tr>
</tbody>
</table>

Details

Only the NAME item is required, but any column attributes accepted by the _setColumnAttributes method can be passed in this list. If no type is specified, numeric is the default.

Data values for the computed column can be assigned or queried using SCL code. The computed column is added at the end of the row in the data vector. The viewer, if any, determines if the column should be displayed and how to display it.

Note: The _addComputedColumn method cannot be called from the model’s SCL entry.

_addRow

Adds a pending row
**Syntax**

CALL SEND (object-id, '_addRow');

**Details**

The `_addRow` method is an editing method and is not valid when

- browsing a table or when the NOADD option is specified in the `_setDataset` method, or
- Addition/Duplication Allowed is disabled under the Table Characteristics of Data Form or Data Table.

If there is an attached viewer, you are placed on the pending row.

The `_addRow` method does not actually add the new row to the table. The add is pending until one of the following occurs:

- the row is committed to the table by using
  - the `_commitNewRow` method, or
  - the Commit New Row item on the pop-up menu
- the current row is changed, for example, by vertically scrolling, or using `_setKey` or `_setWhere`
- the frame is ended (if there is an attached viewer),
- a new row is added or copied, or
- a WHERE clause is issued.

To cancel the pending row, use the `_reread` or `_rereadAll` method, or use the Cancel Row Edits item on the pop-up menu.

---

**_commitNewRow**

Commits the pending row to the table

**Syntax**

CALL SEND (object-id, '_commitNewRow');

**Details**

The `_commitNewRow` method writes any pending row to the table assuming data validation is successful. The newly committed row does not become the current row. You are restored to the row you were on before the add or copy rather than on the committed row.
See Also
_addRow, _copyRow, _reread and _rereadAll.

_copyRow
Copies the locked row as a pending row

Syntax
CALL SEND (object-id, '_copyRow');

Details
The _copyRow method is an editing method and is not valid when
- browsing a table or when the NOADD option is specified in the _setDataset
  method, or
- Addition/Duplication Allowed is disabled under the Table Characteristics of Data
  Form or Data Table.

If there is an attached viewer, you are placed on the pending row.
The _copyRow method does not actually add the new row to the table. The add is
pending until
- the row is committed to the table by using
  - the _commitNewRow method, or
  - the Commit New Row item on the pop-up menu
- the current row is changed, for example, by vertically scrolling, or using _setKey
  or _setWhere
- the frame is ended (if there is an attached viewer),
- a new row is added or copied, or
- a WHERE clause is issued.

To cancel the pending row, use the _reread or _rereadAll method, or use the Cancel
Row Edits item on the pop-up menu.

If there is an attached viewer, then you are placed on the pending row. If you are
using the Data Form and the attached viewer is the form editor, then the _copyRow
method copies the currently displayed row. If you are using the Data Table and the
attached viewer is the table editor, then you must lock the row to be copied.

_deleteComputedColumn
Deletes a computed column

Syntax
CALL SEND (object-id, '_deleteComputedColumn, col-name);
**_deleteRow_**

Delete the current row from the table

**Syntax**

```
CALL SEND (object-id, '_deleteRow');
```

**Details**

The `_deleteRow` method is an editing method and is not valid when

- browsing a table or when the NODELETE option is specified in the `_setDataset` method, or
- Deletion Allowed is disabled under the Table Characteristics of Data Form or Data Table.

**CAUTION:**

Deletions cannot be recovered. The `_deleteRow` method deletes rows from the table, not just from the display of the table. You cannot recover the contents of a deleted row. A row must be locked before it can be deleted. Once the deletion is completed, the model no longer has a current row. The `_deleteRow` method sets SYSRC for error, note, and warning conditions.

**_displayColumnLabel_**

Specifies which columns use table column labels as viewer column label text

**Syntax**

```
CALL SEND (object-id, '_displayColumnLabel', col-name-1<...col-name-n>);
```
### _displayColumnName

Specifies which columns use table column names as viewer column label text

#### Syntax

CALL SEND (object-id, '__displayColumnName', col-name-1<,...,col-name-n>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col-name-1&lt;,...,col-name-n&gt;</td>
<td>C</td>
<td>specifies one or more column names that should display the column name from the table as the column label text for the viewer, if any. This is the default behavior. A single value of the quoted string, '<strong>all</strong>', displays column names for all columns.</td>
</tr>
</tbody>
</table>

#### Details

The _displayColumnName method has no effect if you are running the Data Set Data Model class as a stand-alone object.

Passing a value of '__all__' for the col-name-1 argument of the _displayColumnName method causes the model to use the column names from that point on until another call is made to _displayColumnName that passes one or more column names. For example, calling the _displayColumnName method with the '__all__' parameter makes the attached viewer, if any, use the table column labels for the column label text. If a subsequent call were made to _setDataset, the model would continue to use the table column labels for the new table.

### See Also

__displayColumnName__
viewer, if any, use the table column names for the column label text. If a subsequent call were made to _setDataset, the model would continue to use the table column names for the new table.

---

**_erroroffColumn**

**Turns off the error status of the specified column for the current row**

**Syntax**

CALL SEND (object-id, '_erroroffColumn', col-name-1,<...col-name-n>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col-name-1,&lt;...col-name-n&gt;</td>
<td>C</td>
<td>specifies one or more column names that should have the error status turned off. A single value of the quoted string, '_all', turns off the error status for all columns.</td>
</tr>
</tbody>
</table>

**Details**

If the error status has been turned off for all columns, then the model removes the error from the viewer, if any.

**See Also**

_erroronColumn.

---

**_erroronColumn**

**Turns on the error status of the specified column for the current row**

**Syntax**

CALL SEND (object-id, '_erroronColumn', col-name-1,<...col-name-n>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col-name-1,&lt;...col-name-n&gt;</td>
<td>C</td>
<td>specifies one or more column names that should be set in error. A single value of the quoted string, '_all', turns on the error status for all columns.</td>
</tr>
</tbody>
</table>

**Details**

The specified columns are displayed using the column’s error foreground and background colors.
If the error status is turned on for any columns, then the model sets the viewer, if any, in error. This is done to prevent ending from the frame while a row is in error.
You cannot leave the row if any of its columns are in error.
If a column is placed in error using _errorOnColumn, you must remove the error status using _errorOffColumn, the _override method, issue the OVERRIDE command, or select Override from the pop-up menu, if there is a viewer.

See Also

_ errorOffColumn and _ override.

_execCmd

Processes a command

Syntax

CALL SEND (object-id, '_execCmd');

Details

The _execCmd method is called on a model attached to a viewer only after the viewer tries to process the command. If the command is unrecognized by the viewer, it is sent to the model.

Use the _execCmd method when you want the model to process commands. Note that the command is not passed as an argument. Rather, it processes commands that are sent to the object. Commands are processed in the following order:

1. If the command was issued while in a data form object or data table object, the viewer tries to process the command.
2. If the viewer does not recognize the command, the command is sent to the data set data model object.
3. If the data set data model object does not recognize the command, the command is sent to the data set model object.
4. If the data set model object does not recognize the command, the command is returned to the frame.

Note: If the _execCmd method receives a Data Set Model or a Data Set Data Model command, it commits any current row edits or a pending row to the data set and unlocks the currently locked row, if any.

Example

The following example is an SCL program for a FRAME entry containing either a Data Form or Data Table object called VIEWER. This code allows the FORWARD or BACKWARD commands to scroll in VIEWER regardless of which object on the frame is active at the time the command is issued. All other commands are passed to the model using the _execCmd method.
Make sure commands go through the MAIN section and get the object identifier of the viewer in the frame.

INIT:
   control always;
   call send(_frame_,'_get_widget_','viewer',viewid);
   return;

Get the command and process it.

MAIN:
   command=word(1,'u');
   select(command);

If the command is BACKWARD, scroll backward one row in the viewer and clear the command.

   when ('BACKWARD')
      call send(viewid,'_vscroll_','row',-1);
      call nextcmd();

If the command is FORWARD, scroll forward one row in the viewer and clear the command.

   when ('FORWARD')
      call send(viewid,'_vscroll_','row',1);
      call nextcmd();

Otherwise, process the command.

   otherwise
      if command ne _blank_ then
         call send(viewid,'_execCmd');
      end;
   return;

_fetchRow

Reads a row from the table
Data Table Class

Syntax
CALL SEND (object-id, '_fetchRow', row<, eod>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>N</td>
<td>specifies the relative row number to be read. This row becomes the current row.</td>
</tr>
<tr>
<td>eod</td>
<td>N</td>
<td>returns whether the end of the data has been reached</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 end of data has been reached</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 more records are to be read</td>
</tr>
</tbody>
</table>

Details

Note: You do not have to use _fetchRow to read a row if you are using a data form or a data table. Instead, use _getColumnAttribute, _getColumnText, or _getColumnValue.

You cannot call the _fetchRow method until after initial viewer display or until a row has been read in, for example with _lockRow.

The _fetchRow method sets SYSRC for error, note, and warning conditions.

_findRow

Returns the number of the row that meets the find request

Syntax
CALL SEND (object-id, '_findRow', find-request<, startrow>);
<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>find-request</td>
<td>N</td>
<td>specifies the identifier of an SCL list that contains the find request</td>
</tr>
<tr>
<td>startrow</td>
<td>N</td>
<td>specifies the row on which to start the search. The search begins with startrow. If the find is successful, startrow returns the row number of the match, otherwise it returns -1. By default, if no start-row is defined, the search begins with the current row plus 1. If there is no current row, the search begins with row 1.</td>
</tr>
</tbody>
</table>

**Details**

The find request should contain one or more character list items. Each item can use standard WHERE clause syntax and is handled as an additional request.

**Example**

The following example uses the _findRow method and the _repeatFindRow method. This example assumes you have created a frame with a data table named TABLE and two push buttons named BUTTON1 and BUTTON2.

Get the table's object identifier and set the data set.

```
length charval $ 15;

INIT:
  call send(_frame_,'_get_widget_','table',tabid);
  call send(tabid,'_set_dataset_','sasuser.crime');
  return;
```

When the first push button is pressed, find the row that has STATE = 10 or STATEN = "DELAWARE".

```
BUTTON1:
  row=1;
  list=makelist();
  list=insertc(list,'STATE=10');
  list=insertc(list,'or upcase(staten)="DELAWARE",-1);
  call send(tabid,'_find_row_',list,row);
  call send(tabid,'_lock_row_',row);
  call send(tabid,'_getColumnText_','staten',charval);
  call send(tabid,'_get_column_value_','state',numval);
  put row= charval= numval=;
  list=dellist(list);
```
When the second push button is pressed, repeat the find request.

BUTTON2:
    row+1;
    call send(tabid,'_repeat_find_row_',row);
    put row=;
    return;

See Also
    _repeatFindRow.

__getAutosave

Returns the current autosave setting

Syntax
CALL SEND (object-id, '_getAutosave', autosave);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>autosave</td>
<td>N</td>
<td>returns the current autosave value</td>
</tr>
</tbody>
</table>

See Also
    _setAutosave

__getColumnAttribute

Returns a single attribute for a column

Syntax
CALL SEND (object-id, '_getColumnAttribute', col-name, attr-name, attr-value);
### _getColumnAttributes

**Returns the attributes for a column**

**Syntax**

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col-name</td>
<td>C</td>
<td>specifies the name of the column from which to get the attribute</td>
</tr>
<tr>
<td>attr-name</td>
<td>C</td>
<td>specifies the name of the column attribute</td>
</tr>
<tr>
<td>attr-value</td>
<td>C</td>
<td>N</td>
</tr>
</tbody>
</table>

**Details**

The type of attr-value must match the type of the attribute to be returned.

The _getColumnAttribute method is indirectly delegated to the Data Set Model class, so column attributes for the Data Set Model may be included in this list.

To return more than one column attribute, use the _getColumnAttribute method. For a list of valid column attributes, see “_setColumnAttributes” on page 1139.

---

---

### _getColumnAttributes

**Returns the attributes for a column**

**Syntax**

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

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

**Details**

This list must contain at least a single character item with an item name of NAME, the value of which must be a valid column name.

The _getColumnAttributes method is indirectly delegated to the Data Set Model class, so column attributes for the Data Set Model may be included in this list. By default, all attributes are returned. To return only certain column attributes, include the appropriate named items in your list.

To return a single column attribute, use the _getColumnAttribute method. For a list of valid column attributes, see “_setColumnAttributes” on page 1139.

**Example**

This example gets all the column attributes for the column GENDER in the table SASUSER.CLASS. It also sets the minimum and maximum values for the column HEIGHT in the same table. This example assumes you have created a frame with a data table object named TABLE.

Get the table's object identifier and set the data set.
init:
call send(_frame_,'_get_widget_','table',tabid);
call send(tabid,'_set_dataset_','sasuser.class');

Get the column attributes for GENDER.

gattr=makelist();
gattr=setnitemc(gattr,'sex','name');
call send(tabid,'_get_column_attributes_',gattr);
call putlist(gattr,'All attributes for GENDER=',0);

Set the minimum and maximum value for column HEIGHT.

clearlist(gattr);
gattr=setnitemc(gattr,'height','name');
gattr=setnitemn(gattr,48,'minvalue');
gattr=setnitemn(gattr,72,'maxvalue');
call send(tabid,'_set_column_attributes_',gattr);

clearlist(gattr);
gattr=setnitemc(gattr,'height','name');
gattr=setnitemn(gattr,.,'minvalue');
gattr=setnitemn(gattr,.,'maxvalue');
call send(tabid,'_get_column_attributes_',gattr);
call putlist(gattr,'Min & Max values of column HEIGHT=',0);
dellist(gattr);
return;

_getColumnFormattedData

Returns the formatted column data
### Syntax

CALL SEND (object-id, '_getColumnFormattedData', col-name, data);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col-name</td>
<td>C</td>
<td>specifies the name of the column for which the formatted data is requested</td>
</tr>
<tr>
<td>data</td>
<td>C</td>
<td>returns the formatted data of the requested column</td>
</tr>
</tbody>
</table>

### Details

The _getColumnFormattedData method provides an easy way to obtain formatted data for a column in the currently locked row. For example, using the _getColumnValue method on a date column would return the internal date value, but using _getColumnFormattedData would return the formatted date value.

The _getColumnFormattedData method uses the last Data Set Data Vector. You cannot call the _getColumnFormattedData method until you pass a Data Set Data Vector instance to another method of this object, such as the _getRow method.

### _getColumnNumber

**Returns the column number for a column**

### Syntax

CALL SEND (object-id, '_getColumnNumber', col-name, col-num);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col-name</td>
<td>C</td>
<td>specifies the name of the column for which the number is requested</td>
</tr>
<tr>
<td>col-num</td>
<td>N</td>
<td>returns the number for the named column or -1 if the column does not exist</td>
</tr>
</tbody>
</table>

### Details

The assignment of column numbers is independent of whether the column is displayed in a viewer because the number returned is the position of that column inside the table (data set). In other words, you can call _getColumnNumber for hidden, unhidden, and computed columns.
_getCurrentRowNumber

Returns the relative row number of the current row

Syntax

CALL SEND (object-id, '_getCurrentRowNumber', row-num);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row-num</td>
<td>N</td>
<td>returns one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the relative row number of the current row.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-1 if no row is current or if no rows are displayed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a missing value if the row is new</td>
</tr>
</tbody>
</table>

Details

If you have a viewer and the viewer is a data form, a form editor, or a subclass of the form editor, then the value returned for row-num is the currently displayed row regardless of the mode and locking level.

If you have a viewer and the viewer is a data table, a table editor, or a subclass of the table editor, then the value returned for row-num varies based on mode and locking level. If the mode is edit with record level locking, then row-num is the currently locked row. If the mode is browse or edit with member level locking, then the value returned for row-num is the row where the table editor's active cell indicator is located.

If no viewer is attached, the current row is the row most recently read using _getRow or _fetchRow.

If you fetch a row with the _fetchRow method and that row is already locked by another edit session on the data set, then _getCurrentRowNumber returns a -1.

_getColumnText

Returns the text for a character column in the current row

Syntax

CALL SEND (object-id, '_getColumnText', col-name, text);
### _getColumnValue

**Description**

Returns the value for a numeric column in the current row

**Syntax**

```
CALL SEND (object-id, '_getColumnValue', col-name, value);
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col-name</td>
<td>N</td>
<td>C specifies the name of the column for which the value is requested</td>
</tr>
<tr>
<td>value</td>
<td>C</td>
<td>N returns the value for the column</td>
</tr>
</tbody>
</table>

**Details**

You cannot call the `_getColumnValue` method unless you have a current row.

The `_getColumnValue` method sets SYSRC for error, note, and warning conditions.

### _getDatasetAttributes

**Description**

Returns the attributes for the table

**Syntax**

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>N</td>
<td>C specifies the identifier of an SCL list to contain the attributes</td>
</tr>
</tbody>
</table>

**Details**

To query all table attributes, the list should be empty. To query specific attributes, the list should contain named items for attributes of interest.
The _getDatasetAttributes method sets SYSRC for error, note, and warning conditions.

**Example**

In the following example, the Data Set Model class is being used as a stand-alone model, that is, the model class is not being used with a data form or data table object.

This example assumes you have created a FRAME entry with a push button object named BUTTON1:

```
Load the Data Set Model. In this example, the data set model object is being used as a stand-alone object. Therefore, if you are using SCL similar to this with a data form object or data table object, you should omit the LOADCLASS and INSTANCE functions.

INIT:
   datcl=loadclass('sashelp.fsp.
                   datast_m.class');
   datid=instance(datcl);

Set the data set.

   call send(datid, '_set_dataset_
             ,
             'sasuser.class');
   return;

Get all of the data set attributes.

BUTTON1:
   gattr=makelist();
   call send(datid, '_get_dataset_attributes_
            ,
            gattr);
   call putlist(gattr, 'Data Set Attributes=',
               0);

Get the ENGINE data set attribute.

   clearlist(gattr);
   gattr=setnitemc(gattr, ', ', 'engine');
   call send(datid, '_get_dataset_attributes_
            ,
            gattr);
   call putlist(gattr, 'Engine Information=',
               0);
   dellist(gattr);
   return;
```
Terminate the classes that were instantiated.

TERM:
    call send(datid,'_term_');
    return;

_getDatasetName

_Returns the name of the data set opened with the _setDataset method_

Syntax
CALL SEND (object-id, '_getDatasetName', name);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>C</td>
<td>returns the name of the data set opened with the _setDataset method, including any data set options passed to _setDataset as part of the data set name. It returns a blank if no data set is open.</td>
</tr>
</tbody>
</table>

See Also
_setDataset.

_getDisplayedColumnName

_Returns the name of a displayed column_

Syntax
CALL SEND (object-id, '_getDisplayedColumnName', relative-col-num, col-name);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>relative-col-num</td>
<td>N</td>
<td>specifies the number of the displayed column for which the name is desired</td>
</tr>
<tr>
<td>col-name</td>
<td>C</td>
<td>returns the name of the specified column</td>
</tr>
</tbody>
</table>

Details
The _getDisplayedColumnName method is only meaningful if you have a viewer and the viewer is a data table, a table editor, or a subclass of a table editor.
_getDisplayedColumns

Returns a list that contains the names of the displayed columns and the order in which they are displayed.

Syntax

CALL SEND (object-id, '_getDisplayedColumns', list);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| list     | N    | specifies the identifier of an SCL list to contain the names of the currently displayed columns. Each name is a separate item in the list.

_getHiddenColumns

Returns a list that contains the names of the currently hidden columns.

Syntax

CALL SEND (object-id, '_getHiddenColumns', list);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| list     | N    | specifies the identifier of an SCL list to contain the names of the currently hidden columns. Each name is a separate item in the list.

_getMsgLevel

Returns the current message level.

Syntax

CALL SEND (object-id, '_getMsgLevel', error_flag, warning_flag, note_flag);
Argument Type Description

error_flag C returns whether error messages from this class are to be displayed:

‘Y’ error messages are displayed (default)

‘N’ error messages are not displayed

warning_flag C returns whether warning messages from this class are to be displayed:

‘Y’ warning messages are displayed (default)

‘N’ warning messages are not displayed

note_flag C returns whether notes from this class are to be displayed:

‘Y’ notes are displayed (default)

‘N’ notes are not displayed

Details
When error_flag, warning_flag, or note_flag is set to ‘N’, messages of that type will not be issued and will be lost.

The message level that you set (through _setMsgLevel) does not control all messages; it only controls those messages that are generated by the model. Other messages are not affected, for example, “At top.”, which is generated by the viewer.

See Also
_setMsgLevel

_getNumberOfColumns

Returns the number of columns in the table

Syntax
CALL SEND (object-id, '_getNumberOfColumns', numcols);

Argument Type Description

numcols N returns the number of columns in the table

Details
The value returned by this method can be passed to the _setNumberOfColumns method of the Data Set Data Vector class to initialize the data vector.

_getRecordInfo

Returns a list of information about the current row
**Syntax**

CALL SEND (object-id, '_getRecordInfo', info-list);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>info-list</td>
<td>N</td>
<td>specifies the identifier of an SCL list to contain information about the current row. This list may contain the items described in the Information List table following.</td>
</tr>
</tbody>
</table>

**Table 62.2 Information List**

<table>
<thead>
<tr>
<th>Item</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| 'CURROW'   | N    | the absolute row number for the current row. It returns a missing value if the row is new. It returns -1 under the following conditions:  
  • there are no rows in the data set  
  • no rows meet the WHERE clause  
  • the access engine used to read the table does not support absolute row numbers. |
| 'LOCKED'   | C    | the ability to lock the row  
  'Y' unable to obtain a lock on the row  
  'N' able to obtain a lock on the row |
| 'NEW'      | C    | the row is pending  
  'Y' the current row is a pending row  
  'N' the current row is not a pending row |
| 'ISMODIFIED' | C | the row has been modified  
  'Y' data in the current row have been modified, even if the values are the same as before  
  'N' data in the current row have not been modified  
  A row is considered modified if changes to the values are entered interactively or through SCL. The changes can be to any columns, including computed columns. |
| 'DATACHANGED' | C | the data in the row (table or computed columns) have been changed  
  'Y' data in the current row have been changed  
  'N' data in the current row have not been changed |
| 'SELECTED' | C    | the locking status of the current row  
  'Y' the current row is locked  
  'N' the current row is not locked |

**Details**

This method returns information similar to the OBSINFO SCL function.
**_getRow_**

Reads a row from the table into the Data Set Data Vector class

---

**Syntax**

CALL SEND (object-id, '_getRow', datavector-id, row-num);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datavector-id</td>
<td>N</td>
<td>specifies the object identifier of an instance of the Data Set Data Vector class. The values for the specified row are stored in the data vector and can be accessed using Data Set Data Vector class methods.</td>
</tr>
<tr>
<td>row-num</td>
<td>N</td>
<td>specifies the relative row number to be read. It becomes the current row.</td>
</tr>
</tbody>
</table>

**Details**

*Note:* You do not have to use _getRow to read a row if you are using a data form or a data table. Instead, use _getColumnAttribute, _getColumnText, or _getColumnValue.

After calling _getRow, you can use the _getColumnText and _getColumnValue methods of the Data Set Model class or the _getText and _getValue methods of the Data Set Data Vector class to retrieve the values read from the table.

The _getRow method sets SYSRC for error, note, and warning conditions.

**See Also**

 _lockRow_

---

**_getRowNumber_**

Returns the absolute row number for the current row or the specified relative row

---

**Syntax**

CALL SEND (object-id, '_getRowNumber', row<, rel-row>);
### _getRowNumber_

**Argument** | **Type** | **Description**
---|---|---
row | N | returns the absolute row number for the current row, or the relative row number of `rel-row`, if specified. It returns -1 if the absolute row number is not available or there is no current row.
rel-row | N | specifies the relative row number for which the absolute row number is returned. Note that specifying `rel-row` changes the current row to the row referenced by `rel-row`.

**Details**

You cannot call the _getRowNumber method until after initial viewer display or a row has been read in, for example with _lockRow.

### _getViewerAttribute_

**Returns the value of a viewer attribute for a particular column**

**Syntax**

CALL SEND (object-id, '_getViewerAttribute', col-name, attr-name, attr-value1<...attr-value-n>);

**Argument** | **Type** | **Description**
---|---|---
col-name | C | specifies the name of the column for which to get the attribute.
attr-name | C | specifies the name of the viewer attribute to be returned.
attr-value | C | N | specifies the value of the viewer attribute to be returned. Multiple values are returned when the viewer attribute has more than one value. For example, viewer attribute BDRSTYLE has two values, one for the type of border and the second for the area of the window.

**Details**

This method allows you to get a viewer attribute by passing in the column name with the viewer attribute and a variable to hold the value of the viewer attribute. See the _setViewerAttribute method for a list of the attributes that can be returned by _getViewerAttribute.

**Note** The _getViewerAttribute method can only be called from the model's SCL.

### _getWhere_

**Returns the active where clause**
Syntax

CALL SEND (object-id, '_getWhere', list);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>N</td>
<td>specifies the identifier of an SCL list to contain the active WHERE clause, if any. If no WHERE clause is active, then the list is empty.</td>
</tr>
</tbody>
</table>

_gotoAbsoluteRow

Goes to the specified row

Syntax

CALL SEND (object-id, '_gotoAbsoluteRow', row-num);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row-num</td>
<td>C</td>
<td>specifies the row number to display. If the row-num value is greater than the number of rows in the table, the last row in the table is displayed.</td>
</tr>
</tbody>
</table>

Details

The _gotoAbsoluteRow method returns an error when the access engine used to read the table does not support access by absolute row number or when a permanent or temporary WHERE clause is in effect.

The _gotoAbsoluteRow method sets SYSRC for error, note, and warning conditions.

_hideColumn

Hides a list of columns

Syntax

CALL SEND (object-id, '_hideColumn', col-name-1<,...,col-name-n>);
Data Table Class

_arguments_

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col-name-1&lt;,</td>
<td>C</td>
<td>specifies a list of one or more names of columns to hide. A single value of</td>
</tr>
<tr>
<td>...col-name-n&gt;;</td>
<td></td>
<td>the quoted string, '_all', specifies all columns.</td>
</tr>
</tbody>
</table>

_details_

The hidden column is not removed from the table but is removed from the display if you have a viewer.

Note: The _hideColumn method cannot be called from the model’s SCL entry.

_keyCount

Returns the number of rows that meet the current index key

Syntax

CALL SEND (object-id, '_keyCount', rc, count);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rc</td>
<td>N</td>
<td>returns whether the count was successful</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 count was successful</td>
</tr>
<tr>
<td></td>
<td></td>
<td>not 0 otherwise</td>
</tr>
<tr>
<td>count</td>
<td>N</td>
<td>returns the number of rows in the data set that match the current index key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>that is specified with the _setKey method</td>
</tr>
</tbody>
</table>

details

The _keyCount method returns the number of rows that meet the criteria specified by the index key column. The index key column is specified with the last _setKey method used on the table. After the _keyCount method executes, the table is positioned at the first row that meets the criteria defined by the last _setKey method. Use the _getRow or _fetchRow method to read the row.

The _keyCount method sets SYSRC for error, note, and warning conditions.

**CAUTION:**

Using the _keyCount method with composite keys may show a larger number of rows matching the search criteria than you expect. Using a composite key with the _setKey method operates the same way as the _where method only when the condition is EQ. The value returned when the condition is EQ is the same as if the columns specified in the composite key were connected by WHERE conditions that use AND or ALSO.

For all other conditions (GT, GE, LT, or LE) specified with _setKey for a composite key, the composite key columns are concatenated together to form the index key. The number returned by the _keyCount method is the number of rows in the table that satisfy the composite key. For example, if the composite index consists of columns
SEX and AGE and the condition is GT (greater than), the values to search for are concatenated such that key values of \text{F} for SEX and 13 for AGE yield an index key of \text{F13}. Because the search is performed on the concatenated values, some values may meet the search condition that you did not expect, such as SEX of \text{M} and AGE of 11, because the string \text{M11} is considered greater than the string \text{F13}. △

See Also

\_setKey.

\_lockRow

Locks a row in the table

---

Syntax

CALL SEND (object-id, '_lockRow', row-num<, scroll>);)

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row-num</td>
<td>N</td>
<td>specifies the relative row number of the record to lock.</td>
</tr>
<tr>
<td>scroll</td>
<td>C</td>
<td>specifies whether the model should attempt a scroll if the row is not currently displayed in the viewer. 'NOSCROLL' scrolling will not be attempted (default) 'SCROLL' scrolling will be attempted</td>
</tr>
</tbody>
</table>

Details

If \text{SCROLL} is specified for the scroll parameter, the model will signal to the viewer to scroll the table if the row to be locked is not currently displayed in the viewer. If the row is currently displayed, no scroll action takes place.

\_moveColumn

Moves a range of columns

---

Syntax

CALL SEND (object-id, '_moveColumn', start, end, after);
### _overrideOnError

Allows or disallows override for columns in error

**Syntax**

```plaintext
CALL SEND (object-id, '_overrideOnError', override);
```

**Details**

The _overrideOnError method allows or disallows override for columns in error. If override is allowed, calling the _override method enables you to leave a row even when columns are in error.
### _overrideOnRequired_

**Argument Type Description**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>override</td>
<td>C</td>
<td>'Y' allow override on errors (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' do not allow override on errors.</td>
</tr>
</tbody>
</table>

**Syntax**

CALL SEND (object-id, '_overrideOnRequired'<, override>);

### _protectColumn_

**Argument Type Description**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col-name-1 &lt;...col-name-n&gt;</td>
<td>C</td>
<td>specifies one or more column names to protect. A single value of the quoted string, '_all', specifies all columns.</td>
</tr>
</tbody>
</table>

**Syntax**

CALL SEND (object-id, '_protectColumn', col-name-1,<...col-name-n>);

**Details**

If you are using a data form, a form editor, or a subclass of the form editor, the effect of _protectColumn is that you can type into columns that are protected, but the widgets revert to their previous state when you press enter. If you are using a data table, a table editor, or a subclass of a table editor, the effect of _protectColumn is that you cannot type into the column at all.
**_repeatFindRow**

Finds the next row that meets the last find request

---

**Syntax**

CALL SEND (object-id, '_repeatFindRow'<, startrow>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startrow</td>
<td>N</td>
<td>specifies the row to start searching from, including a search of that row. The startrow argument returns the next matching row or --1 if no match. By default, if no start-row is defined, the search begins with the current row plus 1.</td>
</tr>
</tbody>
</table>

**Details**

By default, if no start-row is defined, the search begins with the current row plus 1.

If the end of the table is reached without a match, the last row of the table becomes the current row. If the last _findRow or _repeatFindRow reached the end without a match, the search begins at the beginning of the table.

The _repeatFindRow method uses the find request specified by the last _findRow method call.

For an example of using _repeatFindRow, see _findRow.

**See Also**

Find _findRow

---

**_reread**

Rereads the current row from the table

---

**Syntax**

CALL SEND (object-id, '_reread');

**Details**

If you have edited values in the current row, calling _reread discards all edits made to the current row since the last time edits were committed to the table (data set).

If you have a pending row, the _reread method effectively cancels the add state and returns you to the state of the display prior to the _addRow or _copyRow method calls, without committing the pending row.
_rereadAll

Rereads the displayed row(s) from the table

Syntax
CALL SEND (object-id, '_rereadAll');

Details
When using the _rereadAll method and the viewer is a data form, a form editor, or a subclass of the form editor, _rereadAll is equivalent to _reread. If the viewer is a data table, a table editor, or a subclass of the table editor, all currently displayed rows are reread from the data set.

If you have edited values in the current row, calling _rereadAll discards all edits made to the current row since the last time edits were committed to the table (data set).

If you have a pending row, the _rereadAll method effectively cancels the add state and returns the display to the its state prior to the _addRow or _copyRow method calls, without committing the pending row.

Note: The _rereadAll method cannot be called from the model’s SCL entry.

__saveAs

Saves the table under a new name

Syntax
CALL SEND (object-id, '_saveAs', name<, replace<, col-1,..., col-n>>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>C</td>
<td>specifies the new name to use when saving the table</td>
</tr>
<tr>
<td>replace</td>
<td>C</td>
<td>specifies whether to replace an existing table name:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' does not replace the table (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' replaces the table if it exists</td>
</tr>
<tr>
<td>col</td>
<td>C</td>
<td>specifies one or more column (variable) names separated by commas that specify the columns to be written to the new table. If none are specified, all are written to the new table.</td>
</tr>
</tbody>
</table>

Details
The _saveAs method will write the rows to the table in the same order they were read by the engine. For example, if a WHERE clause or _setKey method has been applied, only those rows meeting the WHERE clause or set key condition are written to the table.
The _saveAs method sets SYSRC for error, note, and warning conditions.

**Example**

In the following example, the Data Set Model class is being used as a stand-alone model, that is, the model class is not being used with a data form or data table object.

This example is written as a stand-alone SCL entry:

```sas
LOADCLASS('SASHHELP.FSP.DATASET_M.
  CLASS');
DATID=INSTANCE(DATCL);

CALL SEND(DATID,'_SET_DATASET_','SASUSER.CLASS');

CALL SEND(DATID,'_SAVE_AS_','WORK.TEST','Y','NAME','AGE','SEX');

RETURN;

TERM:
  CALL SEND(DATID,'_TERM_');
  RETURN;
```

_setsAutosave

Sets a new autosave value
Syntax

CALL SEND (object-id, '_setAutosave', autosave);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>autosave</td>
<td>N</td>
<td>specifies the new autosave value for the table. This value must be greater than 0.</td>
</tr>
</tbody>
</table>

Details

The _setAutosave method specifies how frequently the model automatically saves the table. The autosave value determines how many rows must be changed before an automatic save is performed. By default, the table is saved automatically whenever 25 rows have been modified since the last save.

To check the current autosave parameter value, use the _getAutosave method.

Regardless of the autosave value, you can save the table at any time by using the _save method.

When the table is saved due to the autosave setting, you no longer have a current row. Methods requiring a current row, such as the _copyRow and _deleteRow methods, fail if a new current row is not established after the save by using the _getRow or _fetchRow method.

_save

Saves the table

Syntax

CALL SEND (object-id, '_save');

Details

If you are using data table, a table editor, or a subclass of the table editor, you no longer have a current row when the table is saved. Methods requiring a current row, such as the _copyRow and _deleteRow methods, will fail if a new current row is not established after the save by using the _getRow or _fetchRow method.

If you are using data form, a form editor, or a subclass of the form editor and you issue _save on an existing row, you remain on that row. Using _save on a new row returns the display to its state prior to the _addRow or _copyRow method calls.

To use this method, you must be in edit mode.

The _save method sets SYSRC for error, note, and warning conditions.
_setColumnAttribute

Sets an attribute for a column

Syntax

CALL SEND (object-id, '_setColumnAttribute', col-name, attr-name, attr-value);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col-name</td>
<td>C</td>
<td>specifies the name of the column in which to set the attribute</td>
</tr>
<tr>
<td>attr-name</td>
<td>C</td>
<td>specifies the name of the column attribute to be set</td>
</tr>
<tr>
<td>attr-value</td>
<td>C</td>
<td>N</td>
</tr>
</tbody>
</table>

Details

The type of the attribute value must match the type of the attribute you specify. Note that this method is indirectly delegated to the Data Set Model class, so column attributes for the Data Set Model class can be used for this method.

To specify more than one column attribute, use the _setColumnAttributes method. Note: You cannot change the NAME, TYPE, LENGTH, COMPUTED, or MODIFIED attributes of a column.

A font list is acquired from the SCL FONTSEL function or a call to the _getDataFont method, the _getColumnAttribute method (retrieving the DATAFONT attribute), or a similar method that gets fonts for other objects.

_setColumnAttributes

Sets one or more attributes for a column

Syntax

CALL SEND (object-id, '_setColumnAttributes', list-id);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list-id</td>
<td>N</td>
<td>specifies the identifier of an SCL list that contains the new attributes for the column</td>
</tr>
</tbody>
</table>

Details

This list must contain at least a single character item with an item name of NAME. This value is required and must be a valid column name. Note that this method is
indirectly delegated to the Data Set Model class, so column attributes for the Data Set Model class may be included in this list.

To set a single column attribute, use the _setColumnAttribute method. Note: You cannot change the NAME, TYPE, LENGTH, COMPUTED, or MODIFIED attributes of a column.

A font list is acquired from the SCL FONTSEL function or a call to the _getDataFont method, the _getColumnAttribute method (retrieving the DATAFONT attribute), or a similar method that gets fonts for other objects.

### Table 62.3 Column Attribute Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'NAME'</td>
<td>C</td>
<td>the name of the column; must be eight or fewer characters in length, and must be a valid SAS name. Note that NAME cannot be changed using the _setColumnAttribute or the _setColumnAttributes method.</td>
</tr>
<tr>
<td>'DATFONT'</td>
<td>N</td>
<td>the identifier of an SCL list that contains the font to use when displaying the data</td>
</tr>
<tr>
<td>'LABFONT'</td>
<td>N</td>
<td>the identifier of an SCL list that contains the font to use when displaying the label</td>
</tr>
<tr>
<td>'REQUIRED'</td>
<td>C</td>
<td>'Y' if a value is required for the column 'N' if a value is not required for a column</td>
</tr>
<tr>
<td>'PROTECTED'</td>
<td>C</td>
<td>'Y' if the column is protected 'N' if the column is not protected</td>
</tr>
<tr>
<td>'HIDDEN'</td>
<td>C</td>
<td>'Y' if the column is hidden 'N' if the column is visible</td>
</tr>
<tr>
<td>'MODIFIED'</td>
<td>C</td>
<td>'Y' if the column has been modified 'N' if the column has not been modified. Notes: MODIFIED cannot be changed using the _setColumnAttribute or _setColumnAttributes methods. Although using the MODIFIED method is convenient in some situations, such as where an OVERRIDE occurs, using a check for the MODIFIED attribute is expensive. Instead, column labels should be used to handle modified columns where possible.</td>
</tr>
<tr>
<td>'EBCOLOR'</td>
<td>C</td>
<td>the background color when the column is in error</td>
</tr>
<tr>
<td>'EFCOLOR'</td>
<td>C</td>
<td>the foreground color when the column is in error</td>
</tr>
<tr>
<td>'LBCOLOR'</td>
<td>C</td>
<td>the background color for the column label</td>
</tr>
<tr>
<td>'LFCOLOR'</td>
<td>C</td>
<td>the foreground color for the column label</td>
</tr>
<tr>
<td>'DBCOLOR'</td>
<td>C</td>
<td>the background color for the column data</td>
</tr>
<tr>
<td>'DFCOLOR'</td>
<td>C</td>
<td>the foreground color for the column data</td>
</tr>
<tr>
<td>Item</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>'JUST'</td>
<td>C</td>
<td>the justification format for the column:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'C' (center)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'L' (left)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'R' (right)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' (none)</td>
</tr>
<tr>
<td>'CAPS'</td>
<td>C</td>
<td>the capitalization for the column:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' if the column sets entered values to UPPERCASE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' otherwise</td>
</tr>
<tr>
<td>'INITVALUE'</td>
<td>C</td>
<td>N</td>
</tr>
<tr>
<td>'MINVALUE'</td>
<td>C</td>
<td>N</td>
</tr>
<tr>
<td>'MAXVALUE'</td>
<td>C</td>
<td>N</td>
</tr>
<tr>
<td>'ERROR'</td>
<td>C</td>
<td>'Y' if the column is in error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' if the column is not in error</td>
</tr>
<tr>
<td>'displayLabel'</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' if the column label is not displayed</td>
</tr>
<tr>
<td>'Column_Width'</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>'COMPUTED'</td>
<td>C</td>
<td>'Y' if the column is a computed column</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' if the column is not a computed column</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note that COMPUTED cannot be changed using the _setColumnAttribute or _setColumnAttributes methods</td>
</tr>
<tr>
<td>'TYPE'</td>
<td>C</td>
<td>the type of the column:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'C' for character columns</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' for numeric columns</td>
</tr>
<tr>
<td>'LENGTH'</td>
<td>N</td>
<td>the data length for the column.</td>
</tr>
<tr>
<td>'FORMAT'</td>
<td>C</td>
<td>the format name for the column; it must be appropriate for the column type</td>
</tr>
<tr>
<td>'INFORMAT'</td>
<td>C</td>
<td>the informat name for the column; it must be appropriate for the column type</td>
</tr>
<tr>
<td>'LABEL'</td>
<td>C</td>
<td>the label for the column:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>may be a maximum of 40 characters</td>
</tr>
<tr>
<td>'dataClass'</td>
<td>C</td>
<td>the host control for a column:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COMBOBOX or SPINBOX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If no host control is desired, set to NONE.</td>
</tr>
</tbody>
</table>
### Item Type Description

<table>
<thead>
<tr>
<th>Item</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'dataAttributes'</td>
<td>L</td>
<td>the named list that defines the attributes for a column host control. The attributes are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'ITEMS' specifies a list of valid values for a column. This list will populate the host control.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'READONLY' specifies if a cell value is editable only through host control. Valid values are 'Y'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'TEXTCOMPLETION' specifies if text matching should occur as text is entered. Valid values are 'Y'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'HONORCASE' specifies whether or not text completion (if active) is case sensitive. Valid values are 'Y'</td>
</tr>
<tr>
<td>'editor'</td>
<td>C</td>
<td>the four-level name of the catalog entry that is opened as an editor when the ellipsis (...) button is selected in the cell</td>
</tr>
<tr>
<td>'editorAttributes'</td>
<td>L</td>
<td>the list of items sent to the editor. By default, the table passes the cell contents to the editor as the 'VALUE' list entry item. If you wish to use a different entry item than the cell contents, specify a named item 'VALUE' on the editorAttributes list, and populate it with the desired values.</td>
</tr>
</tbody>
</table>

### Examples

The following three examples illustrate how to place host controls in an active cell.

**Example 1: Spin box list in a cell** This example creates a spin list with items 'one', 'two', 'three', 'four', and 'five' available on the spin list. It also protects the input field.

```plaintext
list = makelist();
rc = insertc(list, 'ONE', -1);
rc = insertc(list, 'TWO', -1);
rc = insertc(list, 'THREE', -1);
rc = insertc(list, 'FOUR', -1);
rc = insertc(list, 'FIVE', -1);
spinlist = makelist();
rc = insertl(spinlist, list, -1, 'ITEMS');
rc = insertc(spinlist, 'Y', -1, 'READONLY');
call notify(object-id, '_setColumnAttribute', column-name, 'dataClass', 'SPINBOX');
call notify(object-id, '_setColumnAttribute', column-name, 'dataAttributes', spinlist);
```

**Example 2: Editor pushbutton in a cell** This example creates an editor pushbutton that will bring up 'SASUSER.EXAMPLE.EXAMPLE.FRAME' when it is selected.

```plaintext
'editor' = 'SASUSER.EXAMPLE.EXAMPLE.FRAME';
call notify(object-id, '_setColumnAttribute', column-name, 'dataClass', 'EDITOR');
call notify(object-id, '_setColumnAttribute', column-name, 'dataAttributes', 'editor');
call notify(object-id, '_setColumnAttribute', column-name, 'editorAttributes', 'EDITOR');
```
Example 3: Combo box and editor pushbutton in a cell  This example creates both a combo box and an editor pushbutton. The combo box has the items 'red', 'green', 'blue', 'gray', 'white', and 'black' available on the drop-down list. The editor pushbutton will bring up 'SASUSER.EXAMPLE.CLRSELECT.FRAME' when it is selected. Text completion is on and is case sensitive.

```plaintext
list = makelist();
rc = insertc(list, 'RED', -1);
rc = insertc(list, 'GREEN', -1);
rc = insertc(list, 'BLUE', -1);
rc = insertc(list, 'GRAY', -1);
rc = insertc(list, 'WHITE', -1);
rc = insertc(list, 'BLACK', -1);
boxlist = makelist();
rc = insertl(boxlist, list, -1, 'ITEMS');
rc = insertl(boxlist, 'Y', -1, 'HONORCASE');
call notify(object-id, '_setColumnAttribute', column-name, 'dataClass', 'COMBOBOX');
call notify(object-id, '_setColumnAttribute', column-name, 'dataAttributes', boxlist);
call notify(object-id, '_setColumnAttribute', column-name, 'editor', 'SASUSER.EXAMPLE.CLRSELECT.FRAME');
```

_setColumnText

Sets the text for a character column in the current row

Syntax

```plaintext
CALL SEND (object-id, '_setColumnText', col-name, text);
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col-name</td>
<td>C</td>
<td>specifies the name of the character column in which to set the text</td>
</tr>
<tr>
<td>text</td>
<td>C</td>
<td>specifies the new text to assign to the character column in the current row</td>
</tr>
</tbody>
</table>

Details

You cannot call the _setColumnText method unless you have a current row. The _setColumnText method sets SYSRC for error, note, and warning conditions.
**_setColumnValue**

Sets the value for a numeric column in the current row

---

**Syntax**

CALL SEND (object-id, '_setColumnValue', col-name, value);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col-name</td>
<td>C</td>
<td>specifies the name of the column in which to set the value</td>
</tr>
<tr>
<td>value</td>
<td>N</td>
<td>specifies the new value to assign to the numeric column in the current row</td>
</tr>
</tbody>
</table>

**Details**

You cannot call the _setColumnValue method unless you have a current row. The _setColumnValue method sets SYSRC for error, note, and warning conditions.

---

**_setDataset**

Sets the data set being accessed by the object

---

**Syntax**

CALL SEND (object-id, '_setDataset', name=<, mode=<, locking=<, option-1<...,option-n>>>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>C</td>
<td>specifies the name of the new data set. The name may include data set options. See SAS Language Reference, Version 6, First Edition for more information.</td>
</tr>
<tr>
<td>mode</td>
<td>C</td>
<td>specifies the open mode: 'BROWSE' or 'EDIT'. The default is 'BROWSE'.</td>
</tr>
<tr>
<td>locking</td>
<td>C</td>
<td>specifies the locking level: 'RECORD' or 'MEMBER'. The default is 'RECORD'. See &quot;control levels&quot; in SAS Component Language: Reference for more information.</td>
</tr>
<tr>
<td>option-1&lt;...option-n&gt;</td>
<td>C</td>
<td>specifies any additional _setDataset method options: 'BRONLY' prevents editing of the table</td>
</tr>
</tbody>
</table>
**Data Table Class**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_setEntry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>'NOADD'</td>
<td></td>
<td>prevents the user from adding new rows to the table</td>
</tr>
<tr>
<td>'NODELETE'</td>
<td></td>
<td>prevents the user from deleting rows in the table</td>
</tr>
</tbody>
</table>

**Details**

The _setDataset method removes all computed columns defined using the _addComputedColumn method. Recursive calls to _setDataset are not allowed.

**Note**

- The _setDataset method cannot be called from the model's SCL entry.
- If you use the data form, form editor, or a subclass of the form editor as the viewer and the columns in the new table (data set) are different from the previous table (data set), you may need to call _refillUsingAttributes on the viewer in order to see the columns.
- Passing a value of "(a blank enclosed in quotes) as the name of the data set will close the current data set.

**_setDisplayedColumns**

Sets the column names to be displayed and the order in which to display them

**Restrictions**

_setDisplayedColumns has no effect on the order in which columns are displayed in a data form, a form editor, or a subclass of a form editor. However, it does effect which columns are displayed.

**Syntax**

CALL SEND (object-id, '_setDisplayedColumns', col-name-1<,...,col-name-n>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col-name-1&lt;,</td>
<td>C</td>
<td>specifies the name(s) of columns to be displayed.</td>
</tr>
<tr>
<td>...col-name-n&gt;</td>
<td></td>
<td>The order in which the names are passed in is the order in which the names will be displayed.</td>
</tr>
</tbody>
</table>

_setDisplayedColumns does not affect the permanent ordering of columns in the table.

**_setEntry**

Sets the name of the DATAFORM catalog entry associated with the model
**Syntax**

CALL SEND (model-id, '_setEntry', entry<, return-code<, replace>>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entry</td>
<td>C</td>
<td>specifies the name of the DATAFORM catalog entry associated with the model</td>
</tr>
<tr>
<td>return-code</td>
<td>N</td>
<td>contains the return code from _setEntry:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 entry does not yet exist and previously specified entry does exist. Result: A DATAFORM catalog entry by the new name is created on the next _writeEntry call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 entry could not be opened for writing; the entry exists but the user does not have WRITE access to it. Result: The DATAFORM catalog entry is successfully read, but calls to the _writeEntry method do not save any customizations to that DATAFORM entry.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 entry does not yet exist and previously specified entry does not exist. Result: A DATAFORM catalog entry by the new name is created on the next _writeEntry call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 entry successfully opened</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-1 invalid library or entry name Result: The model is not changed at all by the _setEntry call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-2 discrepancy between the column information in the DATAFORM catalog entry and that in the model information. This situation can happen if the DATAFORM entry contains a column that is noncomputed and does not currently exist in the model. Result: The new noncomputed column from the entry is not added to the model.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-3 out-of-memory condition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4 there is already a column in the model by the same name as a column in the entry and the two like-named columns are of different type. Result: The new column from the entry is not moved to the model; the existing model column remains unchanged.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-5 both the conditions described above for values of --2 and --4 occurred (discrepancy between the columns, and there is already a column in the model by the same name as a column in entry and the existing columns are of different types)</td>
</tr>
<tr>
<td>replace</td>
<td>C</td>
<td>specifies whether to replace the existing columns with the columns from the new entry:</td>
</tr>
</tbody>
</table>
### _setKey

Sets an index key for retrieving the rows in the table

#### Syntax

```plaintext
CALL SEND (object-id, '_setKey', rc<, keyname<, condition<, scroll<, val-list>>>);
```

#### Details

The _setKey method sets an index key for retrieving the rows in the table. The key name can be a two- or four-level name.

You can use the optional return-code argument to determine if the _setEntry call was successful.

If there is a column in the entry that is not in the model, it will be added to the model only if the new column is a computed column. Otherwise, the new column will not be added (see -2 above).

If there is a column in the entry that has the same name as a column in the model, the column from the entry will be added to the model if the new column and the model column are both of the same type. Otherwise, the new column will not be added (see -4 above).

### _setInitial

Sets the initial values to be used for any newly added row

#### Syntax

```plaintext
CALL SEND (object-id, '_setInitial'<, 'CLEAR'>);
```

#### Details

The _setInitial method sets the column values in the current row as the initial column values to be used for any added rows. If you specify the optional 'CLEAR' parameter, it restores the initial values for all columns to missing values.
<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rc</td>
<td>N</td>
<td>returns 0 if the key was successfully applied, nonzero otherwise</td>
</tr>
<tr>
<td>keyname</td>
<td>C</td>
<td>specifies the name of the key or index to use on the table. The keyname may specify a single or compound index.</td>
</tr>
<tr>
<td>condition</td>
<td>C</td>
<td>specifies the condition to use when comparing the key value:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'EQ' equal to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'GE' greater than or equal to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'GT' greater than</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'LE' less than or equal to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'LT' less than</td>
</tr>
<tr>
<td>scroll</td>
<td>C</td>
<td>specifies whether observations can be retrieved in random order:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'SCROLL' observations can be retrieved in random order (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'NOSCROLL' observations can only be retrieved sequentially</td>
</tr>
<tr>
<td>val-list</td>
<td>N</td>
<td>specifies the identifier of an SCL list that contains values to use in the key. The item name should reflect the appropriate column name, and the item value should be the value that the key value is compared against.</td>
</tr>
</tbody>
</table>

**Details**

The _setKey method enables you to set an active key in an open table to a simple or composite key. It establishes a set of criteria for reading table rows by evaluating the value of the columns against the key value in the rows.

Using a composite key with _setKey operates the same way as the _where method only when the condition is EQ. The value returned when the condition is EQ is the same as if the columns specified in the composite key were connected by WHERE conditions that use AND or ALSO.

For all other conditions (GT, GE, LT, or LE) specified with _setKey for a composite key, the composite key columns are concatenated together to form the index key. The number returned by the _keyCount method is the number of rows in the table that satisfy the composite key. For example, if the composite index consists of columns SEX and AGE and the condition is GT (greater than), the values to search for are concatenated such that key values of F for SEX and 13 for AGE yield an index key of F13. Because the search is performed on the concatenated values, some values may meet the search condition that you did not expect, such as SEX of M and AGE of 11, because the string M11 is considered greater than the string F13.

Once an active key is set through the _setKey method, it remains active until the following conditions are met:

- the table is closed
- another key is set
- the current setting is cleared by passing the rc argument alone to the _setKey method.

The table is automatically positioned at the row that meets the specified criteria. The _setKey method cannot be used in conjunction with a WHERE clause. The _setKey method sets SYSRC for error, note, and warning conditions.
Example

The following example creates an index on the STATE column in the table SASUSER.CRIME. It subsets on STATE values less than 20. In this example, COUNT returns 15. This example assumes you have created a frame with a data table named TABLE.

Create the index on STATE.

INIT:
   dsid=open('sasuser.crime', 'v');
   icreate(dsid, 'state', 'state');
   close(dsid);

Get the table's object identifier and set the data set.

call notify('.', '_get_widget_',
   'TABLE', tabid);
call send(tabid, '_set_dataset_',
   'sasuser.crime');

Subset on STATE values less than 20.

   list=makelist();
   list=setnitemn(list, 20, 'state');
   call send(tabid, '_set_key_', rc,
      'state', 'LT', 'scroll', list);
   call send(tabid, '_keyCount', rc, count);
   put count=;
   return;

TERM:
   dellist(list);
   return;

__setMsg

Displays a message on the message line, the message window, or in the log

Syntax

CALL SEND (object-id, '__setMsg', string);
Argument         Type          Description

_string          C             specifies the message to be displayed on the message line, the message window, or in the log.

Details
The message is displayed on the message line of the attached viewer’s frame unless one or more of the following conditions exist:
- the object is not currently attached to a viewer
- two or more messages have been issued since the last window refresh
- the message level has been set to ignore this type of message.

In the first two cases, the message will be displayed in the SAS log. If the message level has been set to prevent that type of message from being displayed (for example, if the message begins with ERROR:, NOTE:, or WARNING:, and that message type is turned off), then the message will not be displayed.

If you want to display custom error messages one at a time on the message line such as in the case where multiple columns are in error, you can have the last message written to the message line in most situations by setting the _msg system variable to your message text instead of making multiple calls to _setMsg.

See Also
_setMsgLevel

_setMsgLevel
Specifies whether to issue error, warning, and note messages

Syntax
CALL SEND (object-id, '_setMsgLevel', error_flag<, warning_flag><, note_flag>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_flag</td>
<td>C</td>
<td>specifies whether error messages from this class are to be displayed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘Y’ displays error messages (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘N’ does not display error messages</td>
</tr>
<tr>
<td>warning_flag</td>
<td>C</td>
<td>specifies whether warning messages from this class are to be displayed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘Y’ displays warning messages (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘N’ does not display warning messages</td>
</tr>
<tr>
<td>note_flag</td>
<td>C</td>
<td>specifies whether notes from this class are to be displayed:</td>
</tr>
</tbody>
</table>
Details

The message level is set to allow certain messages generated by this class to be ignored. When error_flag, warning_flag, or note_flag is set to 'N', then messages of that type will not be issued and will be lost.

The message level that you set through _setMsgLevel does not control all messages; it only controls those messages that are generated by the model. Other messages are not affected, for example, "At top.", which is generated by the viewer.

_setOpenmode

Changes the open mode and lock mode for the table

Syntax

CALL SEND (object-id, '_setOpenmode', mode<, locking>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>C</td>
<td>specifies the new open mode for the table: 'EDIT' or 'BROWSE'. You cannot set the open mode to 'EDIT' if 'BRONLY' has been specified for _setDataset.</td>
</tr>
<tr>
<td>locking</td>
<td>C</td>
<td>specifies the new locking mode for the table: 'MEMBER' or 'RECORD'. The default is 'RECORD'.</td>
</tr>
</tbody>
</table>

Details

The _setOpenmode method sets SYSRC for error, note, and warning conditions.

_setRow

Sets the values in the current row from data set data vector

Syntax

CALL SEND (object-id, '_setRow', datavector-id);
### Chapter 62

#### Argument Type Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datavector-id</td>
<td>N</td>
<td>specifies the object identifier for an instance of the Data Set Data Vector class that contains the values to set in the row</td>
</tr>
</tbody>
</table>

#### Details

Note: You do not have to use _setRow to set the values for a row if you are using a data form or a data table. Instead, use _setColumnAttributes, _setColumnText, or _setColumnValue.

You must follow a _setRow call with an _updateRow to write the row to the table. Before you call the _setRow method, you should call the _setText or _setValue methods of the Data Set Data Vector class to set the column values to the desired values.

The _setRow method sets SYSRC for error, note, and warning conditions.

### _setSource

Sets the source entry to associate with the model

---

### Syntax

CALL SEND (model-id, '_setSource', source=<, compile>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>C</td>
<td>specifies the source entry to associate with the model:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'star' an asterisk specifies to use the source entry saved in the DATAFORM catalog entry</td>
</tr>
<tr>
<td>compile</td>
<td>C</td>
<td>specifies whether source should be compiled:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' source will be compiled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' source will not be compiled</td>
</tr>
</tbody>
</table>

#### Details

You can set the source argument to 'star' (an asterisk) to indicate to use the source entry saved in the DATAFORM catalog entry. The source entry saved with the DATAFORM entry is the source entry that was specified the last time a data form object or data table object using that DATAFORM entry catalog was saved. Setting source to 'star' has the effect of setting the source entry to this source entry associated with the DATAFORM entry in the attributes screen for the data form object or data table object.

If a previous source entry exists, its DFTERM label will run. Then the _setSource method will run the new source entry’s DFINIT label. For more information, see the Data Set Data Model class.
__setViewerAttribute__

Sets the value of a viewer attribute for a particular column

**Syntax**

CALL SEND (object-id, '_setViewerAttribute', col-name, attr-name, attr-value1<...attr-value-n>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col-name</td>
<td>C</td>
<td>specifies the name of the column for which to set the attribute. Use <em>ALL</em> to apply the attribute to all columns.</td>
</tr>
<tr>
<td>attr-name</td>
<td>C</td>
<td>specifies the name of the viewer attribute to be set.</td>
</tr>
<tr>
<td>attr-value</td>
<td>C</td>
<td>N</td>
</tr>
</tbody>
</table>

Table 62.4 Attributes for _setViewerAttribute_

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Affects</th>
<th>Used for Data Table</th>
<th>Used for Data Form</th>
<th>Number of Values</th>
<th>Valid Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCOLOR</td>
<td>Background color</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>any color valid for the SASCOLOR window</td>
</tr>
<tr>
<td>FCOLOR</td>
<td>Foreground color</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>any color valid for the SASCOLOR window</td>
</tr>
<tr>
<td>BPATTERN</td>
<td>Background pattern</td>
<td>X</td>
<td></td>
<td>1</td>
<td>default</td>
</tr>
<tr>
<td>BDRCOLOR</td>
<td>Border color</td>
<td>X</td>
<td></td>
<td>2</td>
<td>1=top</td>
</tr>
<tr>
<td>BDRSTYLE</td>
<td>Border style</td>
<td>X</td>
<td></td>
<td>2</td>
<td>1=top</td>
</tr>
<tr>
<td>BDRWIDTH</td>
<td>Border width</td>
<td>X</td>
<td></td>
<td>3</td>
<td>1=top</td>
</tr>
<tr>
<td>FONT</td>
<td>Font</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>a valid font list. For more information see Details, below.</td>
</tr>
<tr>
<td>HJ UST</td>
<td>Horizontal justification</td>
<td>X</td>
<td></td>
<td>1</td>
<td>right</td>
</tr>
<tr>
<td>VJ UST</td>
<td>Vertical justification</td>
<td>X</td>
<td></td>
<td>1</td>
<td>right</td>
</tr>
<tr>
<td>LTSOURCE</td>
<td>Light Source</td>
<td>X</td>
<td></td>
<td>1</td>
<td>upper left</td>
</tr>
<tr>
<td>Attribute Name</td>
<td>Affects</td>
<td>Used for</td>
<td>Used For</td>
<td>Number of Values</td>
<td>Valid Values</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------</td>
<td>----------</td>
<td>----------</td>
<td>------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>MARGIN</td>
<td>Margin</td>
<td>X</td>
<td>Data Table</td>
<td>3</td>
<td>1=top</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Form</td>
<td></td>
<td>2=number</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3=fit</td>
</tr>
<tr>
<td>REVERSE</td>
<td>Reverse Video</td>
<td>X</td>
<td>Data Table</td>
<td>1</td>
<td>'Y'</td>
</tr>
<tr>
<td>PROTECT</td>
<td>Protect</td>
<td>X</td>
<td>Data Table</td>
<td>1</td>
<td>'Y'</td>
</tr>
</tbody>
</table>

**Details**

This method allows the user to set a viewer attribute by passing in the column name with the viewer attribute and associated viewer parameters.

A font list is acquired from the SCL FONTSEL function or a call to the _getDataFont method, the _getColumnAttribute method (retrieving the DATAFONT attribute), or a similar method that gets fonts for other objects.

Note: The _setViewerAttribute method can only be called from the model's SCL.

Note: For the border to be visible, you must specify a border width.

**Example**

The following example assumes you have a frame that contains a data table that uses SASUSER.CLASS. When the value for the AGE column is less than 12, the background and foreground color of the AGE column changes to red and white respectively. The following is SCL code for the model:

```sas
INIT:
AGE:
if age < 12 then
  do;
    call send(_viewer_,'_set_viewer_attribute_','age','bcolor','red');
    call send(_viewer_,'_set_viewer_attribute_','age','fcolor','white');
  end;
else
  do;
    call send(_viewer_,'_set_viewer_attribute_','age','bcolor','white');
    call send(_viewer_,'_set_viewer_attribute_','age','fcolor','black');
  end;
return;
```

**_setWhere**

Sets a WHERE clause on the table
Syntax

CALL SEND (object-id, '_setWhere', where-list<, prompt>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>where-list</td>
<td>N</td>
<td>specifies the identifier of an SCL list that contains the WHERE clause(s) to apply to the table. If you specify 'Y' for prompt, you can specify 0 as the list.</td>
</tr>
<tr>
<td>prompt</td>
<td>C</td>
<td>specifies whether the model should display an interactive WHERE window with which the user can create WHERE clauses.</td>
</tr>
</tbody>
</table>

Details

The _setWhere method imposes one or more sets of conditions that rows in the table must meet in order to be displayed. Rows that do not satisfy the specified conditions cannot be displayed or edited.

If you use the _addRow or _copyRow command to add a new row and enter values that do not meet the WHERE conditions, the row cannot be displayed or edited once it is written to the table while the WHERE clause is in effect.

An empty list or a list with the single item 'CLEAR' clears the current WHERE clause.

The _setWhere method cannot be used with _setKey.

The _setWhere method sets SYSRC for error, note, and warning conditions.

If you augment a WHERE clause, the SCL list containing the augmentation must begin with ALS O.

If you use the interactive WHERE window to specify a WHERE clause, the WHERE clause is automatically applied when the user exits from the WHERE window. For more information on the WHERE window, see SAS Guide to the SQL Query Window: Usage and Reference, Version 6, First Edition, or online, open the SQL Query Window and select Help.

Example

The following example assumes you have created a frame with a data table named TABLE and a push button object named BUTTON1.

Get the table's object identifier and set the data set.

```
INIT:
   call notify('.', '_get_widget_', 'TABLE', tabid);
   call send(tabid, '_set_dataset_', 'sasuser.class');
   return;
```
Apply the WHERE clause.

BUTTON1:
listid=makelist();
listid=insertc(listid, "Sex='M'", -1);
listid=insertc(listid, "and weight > 50", -1);
call putlist(listid, 'WHERE clause', 0);
call send(tabid, '_set_where_', listid);
clearlist(listid);

Get the WHERE clause

listid=setnitemc(listid, ' ', 'WHERE_LIST');
call send(tabid, '_get_dataset_attributes_',
        listid);
call putlist(listid, 'Where List', 0);
dellist(listid);
return;

_sort

Sorts the table by one or more columns using one or more options

Syntax

CALL SEND (object-id, '_sort', column(s)<, /option-1<,...option-n>>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>column(s)</td>
<td>C</td>
<td>specifies one or more columns by which the table is to be sorted. Columns are separated by blanks. The order of the column names reflects the order in which to sort. The column name should be preceded by the keyword DESCENDING when the column is to be sorted in descending order. Ascending order is assumed unless otherwise specified. You may specify as many columns as you like.</td>
</tr>
<tr>
<td>option-1&lt;,...option-n&gt;</td>
<td>C</td>
<td>specifies a string that contains one or more sort options to use for the sort. The sort option string must have a slash (/) at the beginning of the string. Sort options are separated by blanks within the string. You may specify as many sort options as you like. Here are some sample option strings: '/noduplicates' '/noduplicates nodupkey'</td>
</tr>
</tbody>
</table>
You can use the options in the following list, depending on your operating system:

- DIAG
- EQUALS
- FORCE
- LEAVE
- LIST MESSAGE
- NODUPKEY
- NODUPLICATES
- NOEQUALS
- OUTPUT
- REVERSE
- SORTSEQ=ASCII | EBCDIC | DANISH | FINNISH | NATIONAL | NORWEGIAN | SWEDISH
- SORTSIZE
- SORTWKNO
- TAGSORT
- TRANTAB

**Details**

SAS views cannot be sorted in place. To sort views, you must specify an output table.

If you specify the same table as the output table and the input table, then the _sort method treats this as if no output table were specified. It attempts to sort the table in place. To sort the table in place requires that the table be set to edit mode and that it must be able to be opened in member level locking.

The _sort method uses the sorting program that the SAS System supports for your operating system.

The _sort method sets SYSRC for error, note, and warning conditions.

When a permanent or temporary WHERE clause is in effect, the _sort method returns an error.

**Example**

The following example assumes you have created a frame with a data table named TABLE:

Get the table's object identifier and set the data set.

```plaintext
INIT:
call notify('.', '_get_widget_', 'TABLE', tabid);
call send(tabid, '_set_dataset_', 'sasuser.class', 'edit');
```

Sort by SEX and NAME in descending order with the NODUPLICATES option.

```plaintext
```
call send(tabid, '_sort_', 'sex descending name', '/noduplicates');
return;

_unhideColumn

Unhides a list of columns

Syntax

CALL SEND (object-id, '_unhideColumn', col-name-1,...col-name-n);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col-name-1,...col-name-n</td>
<td>C</td>
<td>specifies one or more column names to unhide. A single value of the quoted string, '_all', specifies all columns.</td>
</tr>
</tbody>
</table>

Details

When using the _unhideColumn method with a data table, a table editor, or a subclass of the table editor, the unhidden columns are displayed at the end of the row.

With a data form, a form editor, of a subclass of a form editor, _unhideColumn redisplays only those columns that were previously hidden with the _hideColumn method. These columns appear in their original positions.

Note: The _unhideColumn method cannot be called from the model's SCL entry.

_unlockRow

Unlocks the current row

Syntax

CALL SEND (object-id, '_unlockRow');

Details

Once the _unlockRow method executes, there is no longer a current row. This is useful in edit mode because a record is locked when it is read.

The _unlockRow method sets SYSRC for error, note, and warning conditions.
**_unprotectColumn_**

Unprotects one or more columns

---

**Syntax**

CALL SEND (object-id, '_unprotectColumn', col-name-1<...col-name-n>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col-name-1&lt;...col-name-n&gt;</td>
<td>C</td>
<td>specifies one or more column names to unprotect. A single value of the quoted string, '_all', specifies all columns.</td>
</tr>
</tbody>
</table>

**_updateRow_**

Updates the current row in the table

---

**Syntax**

CALL SEND (object-id, '_updateRow');

**Details**

Note: You do not have to use _updateRow to update a row if you are using a data form or a data table. Instead, use _getColumnAttribute, _getColumnText, or _getColumnValue.

The _updateRow method writes the current row values (passed to the _setRow method) to the data set. You should call the _updateRow method after a call to the _setRow method. To use the _updateRow method, you must be in edit mode.

The _updateRow method sets SYSRC for error, note, and warning conditions.

---

**Data Form Viewer Methods**

The following methods support the viewer functionality of the Data Form class.

---

**_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>
<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:

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

5 Select Run to compile the frame’s SCL, then select Run 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.

---

_**_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
**_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);`
### _getDataBackgroundColor

**Description**

Returns the default border color for data cells

**Syntax**

```plaintext
CALL SEND (table-id, '_getDataBorderColor', border-name, 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>

### _getdataBackgroundColor

**Description**

Returns the default background pattern for data cells

**Syntax**

```plaintext
CALL NOTIFY (table-id, '_getDataBackgroundColor', 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 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>

**See Also**

`_setDataBackgroundColor`

### _setDataBackgroundColor

**Description**

Returns the default border color for data cells

**Syntax**

```plaintext
CALL SEND (table-id, '_setDataBackgroundColor', 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: TOP, LEFT, RIGHT, BOTTOM</td>
</tr>
</tbody>
</table>

`getDataBorderStyle`  
Returns the default border style for data cells  

**Syntax**  
`CALL SEND (table-id, '_getDataBorderStyle', 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>
</tbody>
</table>

<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>

`getDataBorderWidth`  
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);

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

### _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>
</table>
| font-list-id| N    | specifies the identifier of an SCL list to contain the font attributes

**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.
__get_data_Hjust__

Returns the default horizontal justification for data cells

**Syntax**

CALL SEND (table-id, '_get_data_Hjust', 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>'LEFT'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'CENTER'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'RIGHT'</td>
</tr>
</tbody>
</table>

__get_data_LightSource__

Returns the default light source for data cells

**Syntax**

CALL NOTIFY (table-id, '_get_data_LightSource', 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>returns the data 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>

__get_data_Margin__

Returns the default margin for data cells

**Syntax**

CALL SEND (table-id, '_get_data_Margin', margin-name, margin-value, margin-unit);
**_getGridColor_**

Returns the grid line color

**Syntax**

```
CALL SEND (table-id, _getGridColor', 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>None</td>
</tr>
</tbody>
</table>

**_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>None</td>
</tr>
</tbody>
</table>

**Note:** See “Units of Measure” on page 1817 for a complete list.

**Where ...**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>None</td>
</tr>
<tr>
<td>N</td>
<td>None</td>
</tr>
<tr>
<td>C</td>
<td>None</td>
</tr>
</tbody>
</table>
### _getGridStyle

Returns the grid line style

---

**Syntax**

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

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

### _getGridWidth

Returns the grid line width

---

**Syntax**

```plaintext
CALL SEND (table-id, '_getGridWidth', 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>returns the line width in width-unit values</td>
</tr>
<tr>
<td>width-unit</td>
<td>C</td>
<td>returns the unit of measure. See &quot;Units of Measure&quot; on page 1817 for a complete list.</td>
</tr>
</tbody>
</table>

### _getHeldColumns

Returns the currently held columns
**Syntax**

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

Where ...

<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 to contain the coordinates of the first held column. See “Coordinate Lists” on page 1817.</td>
</tr>
<tr>
<td>end-col</td>
<td>N</td>
<td>specifies the identifier of an SCL list to contain the coordinates of the last held column. See “Coordinate Lists” on page 1817.</td>
</tr>
</tbody>
</table>

**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 ...

<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 to contain the coordinates of the first held row. See “Coordinate Lists” on page 1817.</td>
</tr>
<tr>
<td>end-row</td>
<td>N</td>
<td>specifies the identifier of an SCL list to contain the coordinates of the last held row. See “Coordinate Lists” on page 1817.</td>
</tr>
</tbody>
</table>

**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>);
Table 62.1

<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:</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>'COLUMN'</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...

<table>
<thead>
<tr>
<th>the table scrolls horizontally...</th>
</tr>
</thead>
<tbody>
<tr>
<td>'PAGE'</td>
</tr>
<tr>
<td>'HALF'</td>
</tr>
<tr>
<td>'MAX'</td>
</tr>
<tr>
<td>'COLUMN'</td>
</tr>
</tbody>
</table>

### See Also

`_setHscroll` and `_hscroll`

---

### `_getLabelBackgroundColor`

Returns the default background color for label cells

### Syntax

```call send (table-id, '_getLabelBackgroundColor', color-name);
```

### Table 62.2

<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 background color name</td>
</tr>
</tbody>
</table>

---

### `_getLabelBackgroundPattern`

Returns the default background pattern for label cells
Syntax

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

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| pattern    | C    | returns the label background pattern:  
|            |      | ‘SOLID’ | ‘100%’ |
|            |      | ‘75%’   |
|            |      | ‘50%’   |
|            |      | ‘25%’   |
|            |      | ‘TRANSPARENT’ | ‘0%’ |

See Also

_getLabelBackgroundColor

_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>
</table>
| border-name       | C    | specifies the desired border section: 
|                   |      | ‘TOP’  
|                   |      | ‘LEFT’ 
|                   |      | ‘RIGHT’ 
|                   |      | ‘BOTTOM’ |
| color-name        | C    | returns the color name |

_getLabelBorderStyle

Returns the default border style for label cells
**Syntax**

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

Where ...

<table>
<thead>
<tr>
<th>Field</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>style</td>
<td>C</td>
<td>returns the border style:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘SOLID’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘DOTTED’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘DASHED’</td>
</tr>
</tbody>
</table>

**getLabelBorderWidth**

Returns the default border width for label cells

**Syntax**

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

Where ...

<table>
<thead>
<tr>
<th>Field</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>returns the border width in width-unit values</td>
</tr>
<tr>
<td>width-unit</td>
<td>C</td>
<td>returns the unit of measure. See &quot;Units of Measure&quot; on page 1817 for a complete list.</td>
</tr>
</tbody>
</table>

**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);
### `_getLabelLightSource`

*Returns the default light source for label cells*

#### Syntax

```sql
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>returns 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>

### `_getLabelMargin`

*Returns the default margin for label cells*

#### Syntax

```sql
CALL SEND (table-id, '_getLabelMargin', margin-name, margin-value, margin-unit);
```
### _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>C</td>
<td></td>
</tr>
</tbody>
</table>

### _getLabelVjust

Returns the default vertical justification for label cells

**Syntax**

```call send (table-id, '_getLabelVjust', just);```

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>just</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

- **just**: 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>C</td>
<td></td>
</tr>
</tbody>
</table>

---

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

```sql
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

```sql
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
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 “Coordinate Lists” 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 “Coordinate Lists” 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)
          START_COLUMN=(1=1)
          END_ROW=(1=4)
          END_COLUMN=(1=2))
```
The text is as follows:

```plaintext
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);
```
Data Table Class

$getViscol$

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.

$getToprow$

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.
__getVisrow

Returns the number of rows that are currently visible

Syntax

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

Where ... Type Description
numrows N returns the number of visible rows
partial-rows N returns the number of partially visible rows. The only possible values are 0 or 1.

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>);

Where ... Type Description
unit C returns the scrolling unit:
  'PAGE'
  'HALF'
  'MAX'
  'ROW'
num-units N returns the number of units to scroll

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 ...  Type  Description
row            N  specifies the identifier of an SCL list that contains the row coordinates. See “Coordinate Lists” on page 1817.
col            N  specifies the identifier of an SCL list that contains the column coordinates. See “Coordinate Lists” on page 1817.
upper-left     N  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:
                1  position the cell in the upper-left corner of the table
                0  if the cell is already visible, don't move it; otherwise, position it in the upper-left corner of the 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
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>);
Data Table Class

<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>'COLUMN'</td>
</tr>
<tr>
<td>num-units</td>
<td>N</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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**

_popup and _setHscroll

_popup

Displays the run-mode pop-up menu

----------

**Syntax**

CALL SEND (table-id, 'popup'fill-lst, selection);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fill-lst</td>
<td>N</td>
<td>specifies the identifier of an SCL list that contains the pop-up menu items that will be appended to the default pop-up menu</td>
</tr>
<tr>
<td>selection</td>
<td>N</td>
<td>returns the selected pop-up menu item</td>
</tr>
</tbody>
</table>

**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 62.5 on page 1186. Items that are not specified in the options list use the default value.</td>
</tr>
</tbody>
</table>

Table 62.5 _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>'SCREEN'</td>
<td>print the current screen only (default)</td>
</tr>
<tr>
<td>'ALL'</td>
<td>print all pages</td>
</tr>
<tr>
<td>'MAJOR'</td>
<td>specifies the order in which to print multiple pages:</td>
</tr>
<tr>
<td>'ROW'</td>
<td>print complete rows before printing the next row (default)</td>
</tr>
<tr>
<td>'COLUMN'</td>
<td>print complete columns before printing the next 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>);

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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 62.6 on page 1187. 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.</td>
</tr>
</tbody>
</table>

Table 62.6 _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);

<table>
<thead>
<tr>
<th>Where</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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 62.6 on page 1187. 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.</td>
</tr>
</tbody>
</table>

**Table 62.7 _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

___

_releaseColumn

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);

Where ...  Type  Description

| start-col | N      | specifies the identifier of an SCL list that contains the coordinates of the first column to highlight. See "Coordinate Lists" on page 1817. |
| end-col   | N      | specifies the identifier of an SCL list that contains the coordinates of the last column to highlight. See "Coordinate Lists" on page 1817. |

_selectRow

Highlights the given row range

Syntax

CALL SEND (table-id, '_selectRow', 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 highlight. See "Coordinate Lists" on page 1817. |
| end-row   | N      | specifies the identifier of an SCL list that contains the coordinates of the last row to highlight. See "Coordinate Lists" on page 1817. |

_setActiveCell

Sets the current cell

Syntax

CALL SEND (table-id, '_ setActiveCell', row, col);
Where ... | Type | Description
--- | --- | ---
row | N | specifies the identifier of an SCL list that contains the row coordinates. See “Coordinate Lists” on page 1817.

col | N | specifies the identifier of an SCL list that contains the column coordinates. See “Coordinate Lists” on page 1817.

**Details**

The `_setActiveCell` method does not make the active cell visible; see `_gotoCell` in this class.

---

**_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 62.8 on page 1191.

---

**Table 62.8** 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. In the data table, table editor, or subclasses of the table editor with Data Set Data Model, the EDIT attribute is set to &quot;Y&quot; by default.</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) that can be used to assist data entry in the cell.</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. See &quot;Multiple Selections&quot; on page 1815 for more information.</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. If this value is set to 'Y', then cursor tracking is turned on for the data table.</td>
</tr>
<tr>
<td>'RESIZE_COLUMNS'</td>
<td>C</td>
<td>'N'</td>
<td>indicates the ability to resize column widths interactively. If this value is set to 'Y', then cursor tracking is turned on for the data table.</td>
</tr>
<tr>
<td>'RESIZE_rowLABELS'</td>
<td>C</td>
<td>'N'</td>
<td>indicates the ability to resize row label cell widths interactively. If this value is set to 'Y', then cursor tracking is turned on for the data table.</td>
</tr>
<tr>
<td>'RESIZE_ROWS'</td>
<td>C</td>
<td>'N'</td>
<td>indicates the ability to resize row heights interactively. If this value is set to 'Y', then cursor tracking is turned on for the data table.</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. See &quot;Label Selection Mode&quot; on page 1816 for more information.</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. See &quot;Column Selection Mode&quot; on page 1816 for more information.</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>
Data Table Class

## Attribute Details

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'SELECT_rowLABELS'</td>
<td>C</td>
<td>'N'</td>
<td>indicates whether a selection of row label will be expanded into a selection of the entire row. See also &quot;Label Selection Mode&quot; on page 1816 for more information.</td>
</tr>
<tr>
<td>'SELECT_ROWS'</td>
<td>C</td>
<td>'N'</td>
<td>indicates whether a selection of any cell, except column labels, will be expanded into a selection of the entire row containing the cell. See &quot;Row Selection Mode&quot; on page 1816 for more information.</td>
</tr>
<tr>
<td>SELECT_TITLES</td>
<td>C</td>
<td>'N'</td>
<td>indicates whether a selection of a title label will be expanded. See &quot;Label Selection Mode&quot; on page 1816 for more information.</td>
</tr>
<tr>
<td>'SELECTIONS'</td>
<td>C</td>
<td>'Y'</td>
<td>indicates the ability to make selections.</td>
</tr>
<tr>
<td>'SINGLE_CLICK'</td>
<td>C</td>
<td>'N'</td>
<td>indicates whether a single click event drives the data table's object label.</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);
```
**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);

Where ... Type Description
border-name C specifies the desired border section:
       'TOP'
       'LEFT'
       'RIGHT'
       'BOTTOM'

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

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);

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

__setDataFont

Sets the default font for data cells

Syntax

CALL SEND (table-id, '_setDataFont', font-list-id);
### _setDataLightSource

Specifies the light source for data cells

#### Syntax

```plaintext
CALL NOTIFY (table-id, '_setDataLightSource', 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 identifier of an SCL list that contains the font list</td>
</tr>
</tbody>
</table>

### _setDataHjust

Sets the default horizontal justification for data cells

#### Syntax

```plaintext
CALL SEND (table-id, '_setDataHjust', 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: 'LEFT', 'CENTER', 'RIGHT'</td>
</tr>
</tbody>
</table>

### 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.
### _setDataMargin

Sets the default margin for data cells

---

### Syntax

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

### Where ... Type Description

<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 data 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>

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

| margin-value | N | specifies the size of the margin in margin-unit values |
| margin-unit  | C | specifies the unit of measure. See "Units of Measure" on page 1817 for a complete list. |

### 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);

### Where ... Type Description

---
Data Table Class

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>just</td>
<td>C</td>
</tr>
<tr>
<td>_setGridColor</td>
<td>specifies the justification type: 'TOP', 'MIDDLE', 'BOTTOM'</td>
</tr>
<tr>
<td>_setGridStyle</td>
<td>C</td>
</tr>
<tr>
<td>_setGridWidth</td>
<td>C</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>
<tbody>
<tr>
<td>style</td>
<td>C</td>
<td>specifies the grid line style: 'SOLID', 'DOTTED', 'DASHED'</td>
</tr>
</tbody>
</table>

_**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 &quot;Units of Measure&quot; 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

_releaseColumn

_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 &quot;Coordinate Lists&quot; 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 &quot;Coordinate Lists&quot; 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>);
### Chapter 62

<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' (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'MAX'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'COLUMN'</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 _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>
</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 &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.
_setLabelColor

Sets the default text color for label cells

Syntax

CALL SEND (table-id, '_setLabelColor', 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>

_setLabelFont

Sets the default font for label cells

Syntax

CALL SEND (table-id, '_setLabelFont', 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 that contains the font list</td>
</tr>
</tbody>
</table>

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 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 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>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>
<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' 'TOP' 'LEFT' 'RIGHT' '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.

---

**_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' 'MIDDLE' 'BOTTOM'</td>
</tr>
</tbody>
</table>

---

**_setLeftcolumn**

Sets the left-most scrollable column

---

**Syntax**

```
CALL SEND (table-id, '_setLeftcolumn', col);
```
### _setMaxcol

Sets the maximum number of table columns

**Syntax**

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

<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 that contains the column coordinates</td>
</tr>
</tbody>
</table>

**Details**

This method is not valid for multidimensional tables.

### _setMaxrow

Sets the maximum number of table rows

**Syntax**

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

<table>
<thead>
<tr>
<th>Where ...</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max</td>
<td>N</td>
<td>specifies the maximum number of columns that can be displayed by scrolling right. A negative number makes the table horizontally dynamic.</td>
</tr>
</tbody>
</table>

**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);

Where ... Type Description
threshold N specifies the column conform threshold in integer values in the range 0 to 99

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);

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.

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:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'PAGE (default)'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'MAX'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'HALF'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'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

_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>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