Overview

The Data Set Data Model class is part of a collection of classes that combine to create the Data Form and Data Table classes. These classes work together to enable you to:
- view and edit SAS data files
- customize the functionality of these classes for your own application development needs.

Parent:

sashelp.fsp.table_m.class

Class:

sashelp.fsp.Data_m.class

Using the Data Set Data Model Class

Using SCL for Data Validation with the Data Set Data Model Class

The Data Set Data Model class allows you to use SCL to perform data validation in your data entry application, as well as control other aspects of your application.

Accessing the Column Name of the SCL Column Label

You can use the SCL variable, _columnName, in the model SCL for data form objects and data table objects to get access to the name of the column for which the SCL
column label is being run. In the DFINIT, DFTERM, INIT, MAIN, and TERM labels, the value for _columnName is _blank. When a column has been modified and the label for that column is driven for the model, the value contained in the SCL variable called _columnName is the name of the column. The _columnName variable must be specified as a character.

**Warning: SAS guidelines prohibit nesting this deep Example**

The following example assumes you have created a frame with a data form. The table used contains columns, COLUMN1- COLUMN5, which are all numeric. The following SCL code is the model SCL for the data form.

```sas
length _column_name_ $8;

column1: column2: column3:
column4: column5:
  call send(_self_, '_getColumnValue,' _column_name_, value);
  if (value > 9) then
    call send(_self_, '_erroronColumn,' _column_name_);
  else call send(_self_, '_erroroffColumn,' _column_name_);
return;
```

**Assigning an SCL Entry to an Object**

You can use the _setSource method to assign an SCL catalog entry to an instance of the Data Set Data Model class. The SCL entry must have been previously compiled to assign it for use on a Data Set Data Model object. The variable types in the compiled entry must match the types of the variables displayed in your data entry application.

You can compile your SCL entry when you assign it to a Data Set Data Model object using the optional parameter on the _setSource method. Compiling your SCL entry in this way helps you identify potential problems in your data entry application through compiler warnings and error messages. Note: If you use the Data Form or Data Table class and you specify an SCL entry via the attribute windows, the _setSource method as well as the methods to drive the individual labels are run for you automatically.

**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 labelled sections:

1. **DFINIT**: an initialization phase before any rows are displayed, marked by the label DFINIT. This labelled 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 to another row, if the current row has been modified. 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

**Summary of SCL Label Running**

The following SCL labels are run when the viewer is a data form, a form editor, or a subclass of the form editor:
- When _setSource is called, the DFINIT label is run.
- When _term or _setSource to another entry is called, the DFTERM label is run.
- When you scroll onto a row, the INIT label is run and the frame object label and frame MAIN label may also run.
- When you modify a row, the column labels and MAIN label are run, along with the frame object label and frame MAIN label.
- When you scroll off a row, the TERM label is run if the row has been changed.
- When you end from the frame, the TERM label is run if the row has been changed. The DFTERM label and the frame TERM label are also run.

The following SCL labels are run when the viewer is a data table, a table editor, or a subclass of the table editor:
- When _setSource is called, the DFINIT label is run.
- When _term or _setSource to another entry is called, the DFTERM label is run.
- When a row is displayed, the INIT label is run.
- When a row is locked, the INIT label is run, and the frame object label and the frame MAIN label may execute.
- When you modify a row, the column labels and MAIN label are run, along with the frame object label and the frame MAIN label.
- When you unlock a row, the TERM label is run if the row has been changed.
- When you end from the frame, the TERM label is run for any locked rows that were changed, and the frame TERM label is also run.

**SCL Variables**

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 data set data 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. The Data Set Data Model class defines four system variables:

- _SELF_, which contains the object identifier for the instance of the Data Set Data Model class used with the model’s SCL. Note that the following methods cannot be called from the model’s SCL entry:
  - _addComputedColumn
  - _attach
  - _deleteComputedColumn
  - _findRow
  - _repeatFindRow
  - _setData
  - _setDataset
  - _setEntry
  - _setProperties
  - _setSource
  - _select
  - viewer, which contains the object identifier for the viewer attached to the data set data model, if any viewer is available.
  - _msg
- _columnName, which contains the name of the column for which the SCL column label is being run. In the DFINIT, DFTERM, INIT, MAIN, and TERM labels, the value for _columnName is _blank.

---

**Commands for the Data Set Data Model Class**

The following commands are supported by the Data Set Data Model class through the _execCmd method. Note that SCL programs should use the method which corresponds to the command; commands are provided for interactive end-user use. Commands not understood by the Data Set Data Model class are delegated to the Data Set Model class.
ADD

adds a pending row.

Note: The ADD command is not valid when browsing a table or if the NOADD option is specified.

By default, all values in a new row are missing. If an initial value has been stored for the column, the value for a new row contains the initial value for the column.

This command corresponds to the method _addRow.

DROP column<...column-n>>;

excludes one or more columns from the display. See also the _hideColumn method.

DUP<n<row>>

copies the current row to a pending row (in record level locking). You must lock the row you wish to copy.

Note: The DUP command is not valid when browsing a table or if the NOADD option is specified.

By default, the row is duplicated once. To duplicate the same row again, leave the cursor on the command line and execute the DUP command again. Alternatively, you can follow the DUP command with the desired number of copies. Only the last added row is displayed. The rest of the rows are automatically committed to the table. For example, the dup 3 command duplicates the current row three times, committing the first two rows and leaving the third row in pending mode.

In member level locking, the DUP command copies the specified row n times and adds n - 1 rows to the table. The last row is a pending row presented for editing. If no row is specified, the current row is used.

You can select the row to copy by supplying its number as the row argument in the DUP command. To specify the row argument, you must also specify the n argument (the number of times you want the row duplicated). For example, the following command duplicates row 5 two times:

dup 2 5

This command corresponds to the method _copyRow.

CREATE dsname <REPLACE><WITH <ALL | varlist>>

creates a new SAS table using some or all of the columns from the current table. The new table duplicates both the structure and contents of the current table.

If you issue the CREATE command from an attached viewer, then computed columns are not written to the created table. For more information, see the Data Set Model class.

MOVE start end after

moves a range of columns that start with the start column and end with the end column after the after column.

This command corresponds to the method _moveColumn.

PROTECT ON|OFF column<...column-n>

protects a column or range of columns.

This command corresponds to the method _protectColumn.
SHOW column<...column-n>
redisplays dropped or hidden columns.

This command corresponds to the method _unhideColumn.

---

**Default Pop-up Menu**

The pop-up menu displayed by a viewer using the Data Set Data Model class is a combination of the viewer-specific items and the items listed in this section, which the Data Set Data Model class provides. The Data Set Data Model class defines and processes the following pop-up menu items via the _getActions method. Note that items on the pop-up menu may be grayed depending on current option settings and the current state of the model. The viewer is responsible for actually displaying the pop-up menu. Thus, you must have a viewer to have access to the pop-up menu.

The methods used for some of these actions require that you pass parameters. If the method related to an action requires parameters, you must execute your own custom method to specify the parameters to be sent to the method being executed.

**Add Row**
runs the _addRow method to add a pending row.

**Browse/Edit**
runs the _setOpenMode method to alternate between BROWSE mode and EDIT mode.

**Cancel Row Edits**
runs the _reread method to reread the current row from the table. If you have added or copied a row, this item cancels the add or copy without saving the pending row on the table. Otherwise, this item cancels any edits that you have made on the locked row.

**Commit New Row**
runs the _commitNewRow method to add the pending row to the table if you have added or copied a row.

**Copy Row**
runs the _copyRow method to make a pending copy of the locked row in the table.

**Delete Row**
runs the _deleteRow method to delete the locked row from the table.

**Help**
runs the _columnHelpText method to obtain column help from the model.

**Override**
runs the _override method to override required and error fields. This item enables you to override a column that is in error if option settings allow.

**Record/Member**
runs the _setOpenMode method to change alternate between RECORD level locking and MEMBER level locking.
Where
  runs the _setWhere method to open the interactive WHERE window to specify a
  WHERE clause for the data.

Where Clear
  runs the _setWhere method to clear a WHERE clause for the data.

Methods

Methods specific to the Data Set Data Model class are described here. The Data Set
Model class is a delegate of the Data Set Data Model class, so any methods not
contained in the Data Set Data Model class are delegated to the Data Set Model class.

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);
Details

The _deleteComputedColumn method differs from the _hideColumn method in that the column definition is deleted.

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

__deleteRow

Deletes 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

```sql
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, '_all', 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 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.

### 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>);

**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 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>);

**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 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
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>);
### Data Set Data Model Class

<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.

```plaintext
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".

```plaintext
BUTTON1:
  row=1;
  list=makelist();
  list=insertc(list,'STATE=10');
  list=insertc(list,'or upcase(staten)="DELAWARE"');
  list=insertc(list,'or upcase(staten)="DELAWARE"');
  call send(tabid,'_find_row_',list,row);
  call send(tabid,'_lock_row_',row);
  call send(tabid,'_getColumnText_','staten',charval);
  call send(tabid,'_getColumnValue_','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.

_getColumnAttribute

Returns a single attribute for a column

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

Argument | Type | Description
--------- | ---- | -------------------------------
col-name  | C    | specifies the name of the column from which to get the attribute
attr-name | C    | specifies the name of the column attribute
attr-value| C | N | returns the value of the column attribute

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.

_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.

**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.

```plaintext
init:
    call send(_frame_,'_get_widget_','table',tabid);
    call send(tabid,'_set_dataset_','sasuser.class');
```

Get the column attributes for GENDER.

```plaintext
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.

```plaintext
clearlist(gattr);
gattr=setnitemc(gattr,'height','name');
gattr=setnitemn(gattr,48,'minvalue');
gattr=setnitemn(gattr,72,'maxvalue');
call send(tabid,'_set_column_attributes_');
```
Get the minimum and maximum value for column HEIGHT.

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;

_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);

<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 text is requested</td>
</tr>
<tr>
<td>text</td>
<td>C</td>
<td>returns the text of the requested column</td>
</tr>
</tbody>
</table>

Details
You cannot call the _getColumnText method unless you have a current row.

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

**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>C</td>
<td>specifies the name of the column for which the value is requested</td>
</tr>
<tr>
<td>value</td>
<td>N</td>
<td>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.

__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>
<tbody>
<tr>
<td>list</td>
<td>N</td>
<td>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.</td>
</tr>
</tbody>
</table>

__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>
<tbody>
<tr>
<td>list</td>
<td>N</td>
<td>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.</td>
</tr>
</tbody>
</table>

__getMsgLevel

Returns the current message level

Syntax
CALL SEND (object-id, '_getMsgLevel', 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>returns whether error messages from this class are to be displayed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' error messages are displayed (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' error messages are not displayed</td>
</tr>
<tr>
<td>warning_flag</td>
<td>C</td>
<td>returns whether warning messages from this class are to be displayed:</td>
</tr>
</tbody>
</table>
Chapter 59

Argument Type Description

'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

_getRecordInfo

Returns a list of information about the current row

Syntax

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

Argument Type Description

info-list N 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.

Table 59.1 Information List

<table>
<thead>
<tr>
<th>Item</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'CURROW'</td>
<td>N</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• there are no rows in the data set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• no rows meet the WHERE clause</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• the access engine used to read the table does not support absolute row numbers.</td>
</tr>
<tr>
<td>'LOCKED'</td>
<td>C</td>
<td>the ability to lock the row</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' unable to obtain a lock on the row</td>
</tr>
</tbody>
</table>
Data Set Data Model Class

getRow 1013

<table>
<thead>
<tr>
<th>Item</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'N'</td>
<td></td>
<td>able to obtain a lock on the row</td>
</tr>
<tr>
<td>'NEW'</td>
<td>C</td>
<td>the row is pending</td>
</tr>
<tr>
<td>'Y'</td>
<td></td>
<td>the current row is a pending row</td>
</tr>
<tr>
<td>'N'</td>
<td></td>
<td>the current row is not a pending row</td>
</tr>
<tr>
<td>'ISMODIFIED'</td>
<td>C</td>
<td>the row has been modified</td>
</tr>
<tr>
<td>'Y'</td>
<td></td>
<td>data in the current row have been modified, even if the values are the same as before</td>
</tr>
<tr>
<td>'N'</td>
<td></td>
<td>data in the current row have not been modified</td>
</tr>
<tr>
<td>'DATACHANGED'</td>
<td>C</td>
<td>the data in the row (table or computed columns) have been changed</td>
</tr>
<tr>
<td>'Y'</td>
<td></td>
<td>data in the current row have been changed</td>
</tr>
<tr>
<td>'N'</td>
<td></td>
<td>data in the current row have not been changed</td>
</tr>
<tr>
<td>'SELECTED'</td>
<td>C</td>
<td>the locking status of the current row</td>
</tr>
<tr>
<td>'Y'</td>
<td></td>
<td>the current row is locked</td>
</tr>
<tr>
<td>'N'</td>
<td></td>
<td>the current row is not locked</td>
</tr>
</tbody>
</table>

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>);

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-value-1<...attr-value-n>);
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.

_gotoAbsoluteRow

Goes to the specified row

Syntax

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

Argument Type Description
row-num C 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.

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>);

<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 the quoted string, '_all', specifies all columns.</td>
</tr>
<tr>
<td>...col-name-n&gt;</td>
<td></td>
<td></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.

(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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>'NOSCROLL' scrolling will not be attempted (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'SCROLL' scrolling will be attempted</td>
</tr>
</tbody>
</table>

Details

If 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);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>C</td>
<td>specifies the name of the column at the start of the range to be moved</td>
</tr>
<tr>
<td>end</td>
<td>C</td>
<td>specifies the name of the column at the end of the range to be moved</td>
</tr>
<tr>
<td>after</td>
<td>C</td>
<td>specifies the name of the column after which to move the range</td>
</tr>
</tbody>
</table>

Details
The _moveColumn method has no visible effect on columns if the viewer is a data form, a form editor, or a subclass of the form editor. The _moveColumn method does not affect the permanent ordering of columns in the table.

To move a single column, specify its name as start and end.

See Also
_setDisplayedColumns

_override

Overrides required and error fields

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

Details
The _override method has no effect unless OVERRIDE ON ERROR or OVERRIDE ON REQUIRED have been enabled. If override is allowed, calling the _override method enables you to leave a row even when columns are in error.

_overrideOnError

Allows or disallows override for columns in error
Syntax
CALL SEND (object-id, '_overrideOnError', override);

<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>

_overrideOnRequired

Allows or disallows override for required columns

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

<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 required (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' do not allow override on required.</td>
</tr>
</tbody>
</table>

_protectColumn

Protects one or more columns

Syntax
CALL SEND (object-id, '_protectColumn', col-name-1, col-name-n);

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

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 can
not 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
_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 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.

## _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);
Arguments

<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 that 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 59.2  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>Item</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>------</td>
<td>-----------------------------------------------------------------------------</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>'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>‘Y’ if the column label is displayed</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>the width for the column in points. If the value of COLUMN_WIDTH is 0, then column width defaults to format-length EN spaces for character columns and format-length FG spaces for numeric columns.</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>Note that COMPUTED cannot be changed using the _setColumnAttribute or _setColumnAttributes methods</td>
<td></td>
<td></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>
</tbody>
</table>
# Item Type Description

<table>
<thead>
<tr>
<th>Item</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'dataClass'</td>
<td>C</td>
<td>the host control for a column: COMBOBOX or SPINBOX. If no host control is desired, set to NONE.</td>
</tr>
<tr>
<td>'dataAttributes'</td>
<td>L</td>
<td>the named list that defines the attributes for a column host control. The attributes are: 'ITEMS' specifies a list of valid values for a column. This list will populate the host control. 'READONLY' specifies if a cell value is editable only through host control. 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', [...]);
```

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.

call notify(object-id, '_setColumnAttribute', column-name,'editor', 'SASUSER.EXAMPLE.EXAMPLE.FRAME');

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.

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

Argument | Type | Description
---------|------|-----------------------------------
col-name | C | specifies the name of the character column in which to set the text
text | C | specifies the new text to assign to the character column in the current row

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>
### 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 affect which columns are displayed.

**Syntax**

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

**Argument**

<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 the name(s) of columns to be displayed. 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.

### _setKey

Sets an index key for retrieving the rows in the table.

**Argument**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
Syntax

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

<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.

<table>
<thead>
<tr>
<th><strong>Create the index on STATE.</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INIT:</strong></td>
</tr>
<tr>
<td>dsid=open('sasuser.crime', 'v');</td>
</tr>
<tr>
<td>icreate(dsid, 'state', 'state');</td>
</tr>
<tr>
<td>close(dsid);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Get the table's object identifier and set the data set.</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>call notify('.', '<em>get_widget</em>', 'TABLE', tabid);</td>
</tr>
<tr>
<td>call send(tabid, '<em>set_dataset</em>', 'sasuser.crime');</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Subset on STATE values less than 20.</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>list=makelist();</td>
</tr>
<tr>
<td>list=setnitemn(list, 20, 'state');</td>
</tr>
<tr>
<td>call send(tabid, '<em>set_key</em>', rc, 'state', 'LT', 'scroll', list);</td>
</tr>
<tr>
<td>call send(tabid, '<em>keyCount</em>', rc, count);</td>
</tr>
<tr>
<td>put count=;</td>
</tr>
<tr>
<td>return;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>TERM:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>dellist(list);</td>
</tr>
<tr>
<td>return;</td>
</tr>
</tbody>
</table>

**_setMsg**

Displays a message on the message line, the message window, or in the log.
**Syntax**

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>C</td>
<td>specifies the message to be displayed on the message line, the message window, or in the log</td>
</tr>
</tbody>
</table>

**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>
</tbody>
</table>
```markdown
<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 notes from this class are to be displayed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' displays notes (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' does not display notes</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);
```
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.

_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 59.3 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>Attribute Name</td>
<td>Affects</td>
<td>Used for Data Table</td>
<td>Used For Data Form</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>BDRCOLOR</td>
<td>Border color</td>
<td>X</td>
<td>X</td>
<td>2</td>
<td>1=top</td>
</tr>
<tr>
<td>BDRSTYLE</td>
<td>Border style</td>
<td>X</td>
<td>X</td>
<td>2</td>
<td>1=top</td>
</tr>
<tr>
<td>BDRWIDTH</td>
<td>Border width</td>
<td>X</td>
<td>X</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>HJUST</td>
<td>Horizontal Justification</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>right</td>
</tr>
<tr>
<td>VJUST</td>
<td>Vertical Justification</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>right</td>
</tr>
<tr>
<td>LTSOURCE</td>
<td>Light Source</td>
<td>X</td>
<td>1</td>
<td>upper left</td>
<td>lower left</td>
</tr>
<tr>
<td>MARGIN</td>
<td>Margin</td>
<td>X</td>
<td>3</td>
<td>1=top</td>
<td>left</td>
</tr>
<tr>
<td>REVERSE</td>
<td>Reverse Video</td>
<td>X</td>
<td>1</td>
<td>'Y'</td>
<td>'N'</td>
</tr>
<tr>
<td>PROTECT</td>
<td>Protect</td>
<td>X</td>
<td>1</td>
<td>'Y'</td>
<td>'N'</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:

```scl
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','bgcolor','white');
call send(_viewer_, '_set_viewer_attribute_',
  'age','bgcolor','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. 'Y' displays an interactive WHERE window 'N' does not display an interactive WHERE window</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 ALSO.

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>>);
### Column(s)

<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 <code>DESCENDING</code> 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>
</tbody>
</table>

### Option-1<,..,option-n>

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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: <code>'/noduplicates' '/noduplicates nodupkey'</code></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 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.

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.

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