Overview

The Row/Column Data Vector class is used for communication between models and viewers in the Data Form and Data Table classes. The Row/Column Data Vector class is useful to you if you want to do either of the following:

- call certain methods of your own instance of the model class directly, or
- make your own subclass of either the model or the viewer class that calls or overrides certain model methods.

The model methods that use the Row/Column Data Vector class are:

- `_getColumnInfo`
- `_getRowInfo`
- `_getTitleInfo`
- `_setColumnInfo`
- `_setRowInfo`
- `_setTitleInfo`.

The `_getRowInfo`, `_getColumnInfo`, `_setRowInfo`, and `_setColumnInfo` methods are methods of the Form Data Model class. The `_getTitleInfo` and `_setTitleInfo` methods are methods of the Table Data Model class.

The model subclasses of the Form Data Model class and the Table Data Model class, such as the Data Set Data Model class, override these methods to provide functionality specific for their model. Because every model class should be a subclass of either the Table Data Model class or the Table Data Model class, the data vector is used by the above methods for all models.

For more information on the Form Data Model class, see the Form Data Model Class. For more information on the Table Data Model class, see the Table Data Model Class.
Most users of the Data Form and Data Table classes need not be concerned with the Row/Column Data Vector class. In the Data Form and Data Table classes, the Row/Column Data Vector class is used internally.

The Row/Column Data Vector class is used in conjunction with a data model to send and receive information about the way data are configured. For example, the Data Set Data Model class uses an instance of the Row/Column Data Vector class in its _getColumnInfo and _getRowInfo methods to fill in values for the viewer and in its _setColumnInfo and _setRowInfo methods to set values for the viewer.

Note that the Row/Column Data Vector class deals with information about the rows and columns rather than the row or column data themselves. The GET_DATA Data Vector class, on the other hand, deals with information about the data. In the table editor, most items set in the Row/Column Data Vector class are interpreted as row-and-column label cell information. In the form editor, most items set in the Row/Column Data Vector class are interpreted as label widget information.

For more information, see the GET_DATA Data Vector Class.

The _getColumnInfo method, the _getRowInfo method, and the _getTitleInfo method of the model take an instance of a Row/Column Data Vector as an argument. When the model is called, the viewer is responsible for creating that instance of the Row/Column Data Vector class. These model methods then fill in the elements of the Row/Column Data Vector class passed to it. When control is returned to the viewer, the viewer queries the elements set in the row/column data vector passed to the model.

Similarly, the _setColumnInfo method, the _setRowInfo method, and the _setTitleInfo method of the model take an instance of a row/column data vector as an argument. When the model is called with one of these methods, the viewer is responsible for creating that instance of the Row/Column Data Vector class. In these methods calls, the viewer is responsible for filling in the elements of the row/column data vector that it desires to be set in the model. The model then queries the elements set in the row/column data vector it receives and sets values accordingly.

Parent:

sashelp.fsp.object.class

Class:

sashelp.fsp.RCDVec.class

Using the Row/Column Data Vector Class

Calling Methods of the Row/Column Data Vector Class

The Row/Column Data Vector class contains information about one column or one row. The viewers request information about rows or columns by calling the _getRowInfo or _getColumnInfo methods of the model one row or column at a time. Likewise, the viewers set information about rows or columns by calling the _setRowInfo or _setColumnInfo methods of the model one row or column at a time.

The caller of the model’s _getRowInfo, _getColumnInfo, _setRowInfo, or _setColumnInfo method is responsible for creating the instance of the Row/Column Data Vector passed as an argument to these methods. Therefore, if you are overriding the model methods listed above, you need not create your own instance of the Row/Column Data Vector class; one will be passed to you because it has already been created. However, if you are calling any of these methods of the model directly or if
your own viewer class is calling any of these methods on the model, you are responsible for creating your own instance of the Row/Column Data Vector class.

If you are making your own instance of a Row/Column Data Vector class, you must set up the row/column data vector with information about to which row or column method calls the Row/Column Data Vector class refers. After making an instance of the Row/Column Data Vector class, you should call the _setCoordinates method to tell the row/column data vector which row or column the instance is concerned with. If the Row/Column Data Vector you create is passed to the _getRowInfo or _setRowInfo method, the information set on the _setCoordinates method call is interpreted as the current row. If the row/column data vector you create is passed to the _getColumnInfo or _setColumnInfo method, the information set on the _setCoordinates method call is interpreted as the current column.

Example

In the following example, a row/column data vector is created and set up with initial values. The rcdvecid can then be sent to a model on a _setColumnInfo call.

Make an instance of the Row/Column Data Vector class.

```plaintext
rcdvecid = instance(loadclass('sashelp.fsp.
    rcdvec'));
```

Declare the instance to apply to column number 2.

```plaintext
collst = makelist();
cr = insertn(collst, 2, -1);
call send(collst, '_setCoordinates', collst);
```

Set the column's label color to green.

```plaintext
call send(rcdvecid, '_setColor', 'green');
```

Set the column's label to use center horizontal justification.

```plaintext
call send(rcdvecid, '_setHjust', 'center');
```

Note that some of the Row/Column Data Vector methods set or return values that are only applicable for either the form editor or the table editor. By default, it is assumed that the instance of the Row/Column Data Vector class you create can be used by either a form editor or a table editor, and you are free to call any methods of the Row/Column Data Vector class. However, if you know when you make your row/column data vector that it will only be used for either a form editor or a table editor, the Row/Column Data Vector class can do error checking for you by program halting when you call a method that is not supported by the viewer you are using. For more information, see _init in this class.
Where Methods Can be Used

The following table shows which methods of the Row/Column Data Vector class can be used with the Form Editor class, and which can be used with the Table Editor class. The table also shows which methods can be used for columns and which can be used for rows. Those methods that are only useful for columns should not be used in a data vector in the _getRowInfo or _setRowInfo methods. Those methods that are only useful for rows should not be used in a data vector in the _getColumnInfo or _setColumnInfo methods.

Table 87.1 Where Methods Can be Used

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Used in Form Editor?</th>
<th>Used in Table Editor?</th>
<th>Only Useful for Columns</th>
<th>Only Useful for Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>_getAlignment</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getAlignmentChar</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>_getAlignJust</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>_getBackgroundColor</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getBackgroundPattern</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getBorderColor</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getBorderStyle</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getBorderWidth</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getClassName</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getClassProperties</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getColor</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getCoordinates</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getDecimals</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>_getDimension</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getEndOfData</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getFont</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getFormat</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getFormatLength</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getHidden</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getHjust</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getInformat</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getInherit</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getLightSource</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getMargin</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getMeasuring</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getName</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getNameJust</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>_getPrinting</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
## Calling Methods of the Row/Column Data Vector Class

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Used in Form Editor?</th>
<th>Used in Table Editor?</th>
<th>Only Useful for Columns</th>
<th>Only Useful for Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>_getProtect</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getRepresentation</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getResizing</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getReverse</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getSoftEndOfData</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getText</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getType</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>_getVjust</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_getWrapping</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_init</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_setAlignment</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_setAlignmentChar</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>_setAlignJ ust</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>_setBackgroundColor</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_setBackgroundPattern</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_setBorderColor</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_setBorderStyle</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_setBorderWidth</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_setClassName</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_setClassProperties</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_setFont</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_setFormat</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_setFormatLength</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_setHidden</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_setHjust</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_setLightSource</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_setMeasuring</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>_setNameJ ust</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Using the Row/Column Data Vector Class with a Form Editor

You need to make special considerations when using the Row/Column Data Vector class with a form editor. When a form editor displays its data, it calls the _getRowInfo method to query information from the model about the number of rows to display. Then it calls _getColumnInfo for each column in the displayed row to query information from the model about the columns to display. Not all of the items in the Row/Column Data Vector class that you can set on a column-by-column basis make sense to be set for each row in a form display. For example, colors can be set for each column on a call to _getColumnInfo. But colors set for a row on a call to _getRowInfo are used by the table editor but are not used by the form editor. The form editor checks all possible items on the return from the call to the model’s _getColumnInfo method, but it only checks the end-of-data items set by the _setEndOfData and _setSoftEndOfData on the return from the call to the model’s _getRowInfo method.

If you are overriding the model’s _getRowInfo method, consider that any items that you set with the exception of the _setEndOfData and _setSoftEndOfData methods are not honored by the form editor.

Row/Column Data Vector Examples

The first example subclasses the Data Set Data Model class and overrides its _getColumnInfo method. The second example subclasses the Data Set Data Model class and overrides its _getColumnInfo and _getRowInfo methods. Then Row/Column Data Vector methods are used to set each of the items. Finally, in a frame, a form editor and a table editor are created that use the Data Set Data Model subclass.

Overriding the _getColumnInfo Method with a Form Editor

The following is the SCL code for the subclass's _getColumnInfo method used with the form editor.

```scl
getcoli:
  method rcdvecid 8;
  viewerid = getnitemn(_SELF_, 'VIEWERID');
  call super(_SELF_, '_getColumnInfo',
```
rcdvecid;
collst = makelist();
call send(rcdvecid, '_getCoordinates',
        collst);
colnum = getitemn(collst, 1);

Make column 1 have a smaller font.

if colnum eq 1 then do;
    flist = makelist();
    flist = setnitemn(flist, 0, 'OPTIONS');
    flist = setnitemn(flist, 2, 'FAMILY');
    flist = setnitemn(flist, -1, 'CHARSET');
    flist = setnitemn(flist, 60, 'POINTS');
    flist = setnitemn(flist, 1, 'STYLE');
    flist = setnitemn(flist, 9, 'WEIGHT');
    flist = setnitemn(flist, -1, 'WIDTH');
    flist = setnitemn(flist, 1, 'PORTABLE');
    flist = setnitemn(flist, 0.46875, 'RATIO');
    call send(rcdvecid, '_setFont', flist);
end;

Make column 3 use red foreground color.

if colnum eq 3 then do;
    call send(rcdvecid, '_setColor', 'RED');
end;

Make column 4 use green background color.

if colnum eq 4 then do;
    call send(rcdvecid, '_setBackgroundColor',
              'GREEN');
end;

Make column 5 use graphic text class.

if colnum eq 5 then do;
    call send(rcdvecid, '_setClassName',
              'sashelp.fsp.gtext');
oid = loadclass('sashelp.fsp.gtext');
ivl = getniteml(oid, '_iv');
ivl2 = copylist(ivl, 'Y');
rc = setlattr(ivl2, 'UPDATE');
Set rotate to 3 degrees.

```scl
crc = setnitemn(ivl2, 3, 'ROTATE');
call send(rcdvecid, '_setClassProperties', ivl2);
end;
```

Make column 6 use region title label.

```scl
if colnum eq 6 then do;
    call send(rcdvecid, '_setRepresentation', 'REGTITLE');
end;
endmethod;
```

**Overriding the _getColumnInfo Method with a Table Editor**

The following is the SCL code for the subclass's _getColumnInfo method used with the table editor.

```scl
getcoli:
method rcdvecid 8;
    viewerid = getnitemn(_SELF_, 'VIEWERID');
call super(_SELF_, '_getColumnInfo', rcdvecid);
collst = makelist();
call send(rcdvecid, '_getCoordinates', collst);
    colnum = getitemn(collst, 1);

Make column 1 have a smaller font.

```scl
if colnum eq 1 then do;
    flist = makelist();
    flist = setnitemn(flist, 0, 'OPTIONS');
    flist = setnitemn(flist, 2, 'FAMILY');
    flist = setnitemn(flist, -1, 'CHARSET');
    flist = setnitemn(flist, 60, 'POINTS');
    flist = setnitemn(flist, 1, 'STYLE');
    flist = setnitemn(flist, 9, 'WEIGHT');
    flist = setnitemn(flist, -1, 'WIDTH');
    flist = setnitemn(flist, 1, 'PORTABLE');
    flist = setnitemn(flist, 0.46875, 'RATIO');
call send(rcdvecid, '_setFont', flist);
end;
Overriding the _getRowInfo Method with a Table Editor

The following is the SCL code for the subclass’s _getRowInfo method used with the table editor.

```scl
getrowi:
method rcdvecid 8;
  viewerid = getnitemn(_SELF_, 'VIEWERID');
  call super(_SELF_, '_getRowInfo',
    rcdvecid);
rowlst = makelist();
```
call send(rcdvecid, '_getCoordinates', rowlst);
rownum = getitemn(rowlst, 1);

Makerow 1 use red foreground color.

if rownum eq 1 then do;
call send(rcdvecid, '_setColor', 'RED');
end;

Makerow 2 use green background color.

if rownum eq 2 then do;
call send(rcdvecid, '_setBackgroundColor', 'GREEN');
end;

Makerow 3 use bottom justification and set vertical justification to 0.5 inch.

if rownum eq 3 then do;
call send(rcdvecid, '_setDimension', 0.5, 'in');
call send(rcdvecid, '_setVjust', 'BOTTOM');
end;

Makerow 4 use yellow borders.

if rownum eq 4 then do;
alist = makelist();
alist = setnitemc(alist, 'N', 'LABEL_BUTTONS');
call send(viewerid, '_setAttributes', alist);
call send(rcdvecid, '_setBorderColor', 'ALL', 'YELLOW');
call send(rcdvecid, '_setBorderWidth', 'ALL', 0.1, 'in');
end;

Makerow 5 use dashed borders.

if rownum eq 5 then do;
call send(rcdvecid, '_setBorderStyle',
    'ALL', 'DASHED');
call send(rcdvecid, '_setBorderWidth',
    'ALL', 0.1, 'in');
end;

Make row 6 use 0.1 inch borders.

if rownum eq 6 then do;
call send(rcdvecid, '_setBorderWidth',
    'ALL', 0.1, 'in');
end;

Make row 7 use 0.7 inch right margin.

if rownum eq 7 then do;
call send(rcdvecid, '_setMargin', 'RIGHT',
    0.7, 'in');
end;

Make row 8 use reverse video.

if rownum eq 8 then do;
call send(rcdvecid, '_setReverse', 'Y');
end;

Make row 9 be 0.5 inch tall with a 0.4 inch baseline.

if rownum eq 9 then do;
call send(rcdvecid, '_setDimension',
    0.5, 'in');
call send(rcdvecid, '_setAlignment',
    0.4, 'in');
end;

Make row 10 cross all rows.

if rownum eq 10 then do;
call send(rcdvecid, '_setNameJust', -1);
end;

Make row 11 have new text.
if rownum eq 11 then do;
   call send(rcdvecid, '_setText', 'newlab');
end;

endmethod;

Methods

Methods specific to the Row/Column Data Vector class are described here. Inherited methods are described in the Object class.

Dictionary

_getAlignment

Returns the baseline or alignment

Syntax

CALL SEND (data-vector-id, '_getAlignment', align<, unit>);

Argument | Type | Description
---------|------|-------------
align     | N    | returns the baseline or alignment value
unit      | C    | returns the baseline or alignment unit of measure.

Details

The _getAlignment method is valid only if the viewer is a table editor. If the Row/Column Data Vector instance is used in conjunction with the _getRowInfo or _setRowInfo method, the value returned by the _getAlignment method is interpreted as the row's base line. This is the distance from the top of the row to start the text. If you use the row/column data vector in conjunction with the _getColumnInfo or _setColumnInfo method, the value returned by the _getAlignment method is interpreted as the row's alignment. This is the distance from the left of the column to start the text. The following conditions also are true when you use a row/column data vector in conjunction with the _getColumnInfo method or the _setColumnInfo method:

- The alignment value specified by the _setAlignment method specifies how much space is available in which to align the text from the left of the column.
- The alignment justification specified by the _setAlignJust method tells how to justify the text horizontally in the width given by the _setAlignment method.
- The alignment character specified by the _setAlignmentChar method is the character on which you want the data in the column to be aligned.
The number of decimals specified by the _setDecimals method is the number of characters allowed to the right of the alignment character specified by the _setAlignmentChar method if the alignment justification specified by the _setAlignJust method is RIGHT.

Use either the _setDecimals or the _setAlignment method, but not both.

__getAlignmentChar__

Returns the alignment character

**Syntax**

CALL SEND (data-vector-id, '_getAlignmentChar', align-char);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>align-char</td>
<td>C</td>
<td>returns the alignment character</td>
</tr>
</tbody>
</table>

**Details**

The _getAlignmentChar method is valid only if the viewer is a table editor.

The _getAlignmentChar method can only be used for a Row/Column Data Vector instance being passed to the model’s _getColumnInfo or _setColumnInfo method. Its values are not meaningful for row settings.

The alignment character, when specified, is typically a decimal point, ‘.’. This is typically used to line up numeric values.

The alignment value specified by the _setAlignment method specifies how much space is available in which to align the text from the left of the column. The alignment justification specified by the _setAlignJust method tells how to horizontally justify the text in the width given by the _setAlignment method. The alignment character specified by the _setAlignmentChar method is the character on which you want the data in the column to be aligned. The number of decimals specified by the _setDecimals method is the number of characters allowed to the right of the alignment character specified by the _setAlignmentChar method if the alignment justification specified by the _setAlignJust method is RIGHT. The _setDecimals and the _setAlignment methods should not both be used. Instead, only one of these methods should be called.

__getAlignJust__

Returns the alignment justification

**Syntax**

CALL SEND (data-vector-id, '_getAlignJust', just);
Argument | Type | Description
---|---|---
just | C | returns the alignment justification:
  'RIGHT'
  'LEFT'
  'CENTER'

**Details**
The `_getAlignJust` method is valid only if the viewer is a table editor.

The `_getAlignJust` method can only be used for a Row/Column Data Vector instance being passed to the model's `_getColumnInfo` or `_setColumnInfo` method. Its values are not meaningful for row settings.

The alignment value specified by the `_setAlignment` method specifies how much space is available in which to align the text from the left of the column. The alignment justification specified by the `_setAlignJust` method tells how to horizontally justify the text in the width given by the `_setAlignment` method. The alignment character specified by the `_setAlignmentChar` method is the character on which you want the data in the column to be aligned. The number of decimals specified by the `_setDecimals` method is the number of characters allowed to the right of the alignment character specified by the `_setAlignmentChar` method if the alignment justification specified by the `_setAlignJust` method is RIGHT. Use either the `_setDecimals` or the `_setAlignment` method, but not both.

---

**_getBackgroundColor**

Returns the background color

---

**Syntax**

```
CALL SEND (data-vector-id, '_getBackgroundColor', color);
```

---

**Argument | Type | Description**
---|---|---
color | C | returns the background color

**Details**
The `_getBackgroundColor` method is valid if the viewer is either a form editor or a table editor.

---

**_getBackgroundPattern**

Returns the background pattern
### Syntax

CALL SEND (data-vector-id, '_getBackgroundPattern', 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 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 `_getBackgroundPattern` method is valid only if the viewer is a table editor.

---

### `_getBorderColor`

**Returns the border outline color**

#### Syntax

CALL SEND (data-vector-id, '_getBorderColor', area, color);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>area</td>
<td>C</td>
<td>specifies the area to return the border color for:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'RIGHT'  the right side of the cell</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'LEFT'   the left side of the cell</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'TOP'    the top side of the cell</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'BOTTOM' the bottom side of the cell</td>
</tr>
<tr>
<td>color</td>
<td>C</td>
<td>returns the border outline color</td>
</tr>
</tbody>
</table>

### Details

The `_getBorderColor` method is valid only if the viewer is a table editor.

---

### `_getBorderStyle`

**Returns the border style**
**Syntax**

CALL SEND (data-vector-id, '_getBorderStyle', area, style);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>area</td>
<td>C</td>
<td>specifies the area to return the border style for:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'RIGHT'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'LEFT'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'TOP'</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>

**Details**

The _getBorderStyle method is valid only if the viewer is a table editor.

---

**_getBorderWidth**

*Returns the border width*

**Syntax**

CALL SEND (data-vector-id, '_getBorderWidth', area, width, unit);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>area</td>
<td>C</td>
<td>specifies the area to return the border width for:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'RIGHT'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'LEFT'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'TOP'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'BOTTOM'</td>
</tr>
<tr>
<td>width</td>
<td>N</td>
<td>returns the border width</td>
</tr>
<tr>
<td>unit</td>
<td>C</td>
<td>returns the border width unit of measure.</td>
</tr>
</tbody>
</table>

**Details**

The _getBorderWidth method is valid only if the viewer is a table editor.
### _getClassProperties

Returns the label class name

#### Syntax

```plaintext
CALL SEND (data-vector-id, '_getClassName', class);
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>C</td>
<td>returns the label class name</td>
</tr>
</tbody>
</table>

#### Details

The _getClassName method is valid only if the viewer is a form editor. The label class is the name of the widget class the form editor uses to display labels. The instance variables list of this class is controlled with the _getClassProperties and _setClassProperties methods.

### _getClassProperties

Returns the label attributes list

#### Syntax

```plaintext
CALL SEND (data-vector-id, '_getClassProperties', attributes);
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes</td>
<td>N</td>
<td>returns the identifier of an SCL list that contains the label attributes</td>
</tr>
</tbody>
</table>

#### Details

The _getClassProperties method is valid only if the viewer is a form editor. The attributes list is the instance variable list of the label class used by the form editor. For example, if the form editor is displaying its labels with the Text Entry class, the attributes list reflects the instance variables list of the Text Entry class. The class being used by the form editor is controlled with the _getClassName and _setClassName methods.
_getColor

Returns the foreground color

**Syntax**

```
CALL SEND (data-vector-id, '_getColor', color);
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>C</td>
<td>returns the foreground color</td>
</tr>
</tbody>
</table>

**Details**
The `_getColor` method is valid if the viewer is either a form editor or a table editor.

__getCoordinates

Returns the row or column coordinates

**Syntax**

```
CALL SEND (data-vector-id, '_getCoordinates', cur-label);
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cur-label</td>
<td>N</td>
<td>returns the identifier of an SCL list that contains the row or column coordinates.</td>
</tr>
</tbody>
</table>

**Details**
The `_getCoordinates` method is valid if the viewer is either a form editor or a table editor.

The row or column indicates to which row or column this row/column data vector refers. Each row/column data vector deals only with one row or column at a time. If this row/column data vector is used as an argument to the `_getRowInfo` or `_setRowInfo` method, this value is interpreted as the current row. If this row/column data vector is used as an argument to the `_getColumnInfo` or `_setColumnInfo` method, this value is interpreted as the current column. See “Example” on page 1595.

__getDecimals

Returns the number of decimal places
**Syntax**

CALL SEND (data-vector-id, '_getDecimals', decimals);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>decimals</td>
<td>N</td>
<td>returns the number of decimal places used for alignment</td>
</tr>
</tbody>
</table>

**Details**

The `_getDecimals` method is valid only if the viewer is a table editor. You can use the `_getDecimals` method only for a row/column data vector being passed to the model's `_getColumnInfo` or `_setColumnInfo` method. Its values are not meaningful for row settings.

The following conditions also are true when you use a row/column data vector in conjunction with the `_getColumnInfo` method or the `_setColumnInfo` method:

- The alignment value specified by the `_setAlignment` method specifies how much space is available in which to align the text from the left of the column.
- The alignment justification specified by the `_setAlignJust` method tells how to justify the text horizontally in the width given by the `_setAlignment` method.
- The alignment character specified by the `_setAlignmentChar` method is the character on which you want the data in the column to be aligned.
- The number of decimals specified by the `_setDecimals` method is the number of characters allowed to the right of the alignment character specified by the `_setAlignmentChar` method if the alignment justification specified by the `_setAlignJust` method is RIGHT.
- Use either the `_setDecimals` or the `_setAlignment` method, but not both.

---

**_getDimension**

Returns the width or height

**Syntax**

CALL SEND (data-vector-id, '_getDimension', dim<, unit<, add<, fit<, fixed>>>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dim</td>
<td>N</td>
<td>returns the width or height value</td>
</tr>
<tr>
<td>unit</td>
<td>C</td>
<td>returns the width or height unit of measure.</td>
</tr>
<tr>
<td>add</td>
<td>C</td>
<td>returns whether the width or height is added to the margins:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' added</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' not added</td>
</tr>
</tbody>
</table>


### _getDimension

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fit</td>
<td>C</td>
<td>returns whether the width or height is a minimum size:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' minimum size</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' not a minimum size</td>
</tr>
<tr>
<td>fixed</td>
<td>C</td>
<td>returns whether the width or height is fixed by the viewer:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' fixed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' not fixed</td>
</tr>
</tbody>
</table>

**Details**

The _getDimension method is valid only if the viewer is a table editor. If `fit` is set to 'Y', the viewer interprets the width or height as a minimum size. This means that the viewer attempts to measure the data to determine their size, but it never goes below the width or height specified.

If `fixed` is set to 'Y', the viewer does not attempt to adjust the size of the data. Margins are ignored, and the viewer does not attempt to ensure that the row or column is big enough to display all of the label or the data.

### _getEndOfData

**Returns the hard end-of-data status**

**Syntax**

```call send (data-vector-id, '_getEndOfData', end);```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>end</td>
<td>C</td>
<td>returns the hard end-of-data status:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' this is the hard end of data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' this is not the hard end of data</td>
</tr>
</tbody>
</table>

**Details**

The _getEndOfData method is valid if the viewer is either a form editor or a table editor. The hard end of data are related to the soft end of data controlled by the _getSoftEndOfData and _setSoftEndOfData methods.

Both hard and soft end-of-data status are used to notify the viewer that the row or column being requested is past the number of rows or columns that the model has. For example, if there are three columns in a model, on a viewer’s call to _getColumnInfo requesting information about column 4, the model sets the end-of-data status to 'Y'.

The difference between setting the hard end of data and the soft end of data is that with the hard end of data you cannot scroll in the table editor, while with the soft end of data you can scroll. The hard end of data is useful for models with a stable and
known number of rows or columns, while the soft end of data is useful for models with a changing or unknown number of rows or columns.

_getFont

Returns the font

Syntax
CALL SEND (data-vector-id, ’_getFont’, font-list-id);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>font-list-id</td>
<td>N</td>
<td>returns the identifier of an SCL list that contains the font information</td>
</tr>
</tbody>
</table>

Details
The _getFont method is valid if the viewer is either a form editor or a table editor. The list returned by the _getFont method should not be modified directly, but can be sent to other objects that take font lists. Changing items in this list and then sending the changed list to the _setFont method may produce unreliable results.

_getFormat

Returns the format

Syntax
CALL SEND (data-vector-id, ’_getFormat’, format);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>C</td>
<td>returns the format</td>
</tr>
</tbody>
</table>

Details
The _getFormat method is valid only if the viewer is a form editor.

_getFormatLength

Returns the format length
**Syntax**

CALL SEND (data-vector-id, '_getFormatLength', length);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>N</td>
<td>returns the format length</td>
</tr>
</tbody>
</table>

**Details**

The _getFormatLength method is valid if the viewer is either a form editor or a table editor. The value returned by the _getFormatLength method is the maximum number of characters the viewer will display.

---

**_getHidden**

Returns the hidden status

**Syntax**

CALL SEND (data-vector-id, '_getHidden', hidden);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hidden</td>
<td>C</td>
<td>returns the hidden status:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' hidden</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' not hidden</td>
</tr>
</tbody>
</table>

**Details**

The _getHidden method is valid only if the viewer is a form editor.

---

**_getHjust**

Returns the horizontal justification

**Syntax**

CALL SEND (data-vector-id, '_getHjust', just);
Argument | Type | Description
--- | --- | ---
just | C | returns the horizontal justification:
   - 'RIGHT'
   - 'LEFT'
   - 'CENTER'

**Details**
The _getHjust method is valid if the viewer is either a form editor or a table editor.

---

**_getInformat**

*Returns the informat*

**Syntax**
CALL SEND (data-vector-id, '_getInformat', informat);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
informat | C | returns the informat |

**Details**
The _getInformat method is valid only if the viewer is a form editor.

---

**_getInherit**

*Returns the inherit status*

**Syntax**
CALL SEND (data-vector-id, '_getInherit', inherit);
### _getInherit

The `_getInherit` method is valid only if the viewer is a form editor. The default inherit status is 'N'. If inherit is set to 'Y', the colors, fonts, and other attributes specified under the form editor attributes screens are ignored in favor of the attributes set in the specified label class.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inherit</td>
<td>C</td>
<td>returns the inherit status:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' the label class inherits its attributes from the label class rather than from the settings in the form editor attributes screens</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' the label class uses the fonts, colors, and other attributes from the form editor attributes screens</td>
</tr>
</tbody>
</table>

### _getLightSource

Returns the light source for button borders

**Syntax**

```
CALL SEND (data-vector-id, '_getLightSource', 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 light source for button borders:</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>

**Details**

The `_getLightSource` method is valid only if the viewer is a table editor.

### _getMargin

Returns the margin settings

**Syntax**

```
CALL SEND (data-vector-id, '_getMargin', area, margin, unit);
```
## _getMargin

Returns the margin setting for:

- `'RIGHT'` the right side of the cell
- `'LEFT'` the left side of the cell
- `'TOP'` the top side of the cell
- `'BOTTOM'` the bottom side of the cell

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>area</td>
<td>C</td>
<td>specifies the area to return the margin setting for:</td>
</tr>
<tr>
<td>margin</td>
<td>N</td>
<td>returns the margin setting value</td>
</tr>
<tr>
<td>unit</td>
<td>C</td>
<td>returns the margin setting unit of measure.</td>
</tr>
</tbody>
</table>

### Details

The `_getMargin` method is valid only if the viewer is a table editor.

---

## _getMeasuring

Returns the measuring status

### Syntax

CALL SEND (data-vector-id, '_getMeasuring', measuring);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>measuring</td>
<td>C</td>
<td>returns the measuring status:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' the viewer is measuring the data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' the viewer is not measuring the data</td>
</tr>
</tbody>
</table>

### Details

The `_getMeasuring` method is valid only if the viewer is a table editor. Note that the viewer, not the model, sets the measuring status. The model queries the status but does not set it.

A measuring state set to 'Y' informs the model that the viewer is measuring its data to determine how large to make its cells. Similar to the printing status controlled by the `_setPrinting` and `_getPrinting` methods, the models are not usually concerned with this unless the model has an unspecified number of rows or columns. If this is the case, the model needs to check this value when returning information to the viewer in the `_getRowInfo` or `_getColumnInfo` method call so that it does not continue to return rows or columns while measuring is occurring.
_getName

Returns the name of the column

Syntax
CALL SEND (data-vector-id, '_getName', 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 column</td>
</tr>
</tbody>
</table>

Details
The _getName method is valid only if the viewer is a form editor. Use the _getText method if the viewer is a table editor.

_getNameJust

Returns the number of columns for justification

Syntax
CALL SEND (data-vector-id, '_getNameJust', num-columns);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num-columns</td>
<td>N</td>
<td>returns the number of columns under which the column name should be justified:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-1</td>
</tr>
</tbody>
</table>

Details
The _getNameJust method is valid only if the viewer is a table editor. You can use the _getNameJust method only to pass a row/column data vector to the model's _getRowInfo or _setRowInfo method. Its values are not meaningful for column settings. The default value of 1 for num-columns indicates to justify the column label under one column. A value of -1 for num-columns indicates to justify the column label under the entire table. This has the effect of making that column label span the entire table. Values other than 1 and -1 are not supported.
_getPrinting

Returns the printing status

Syntax
CALL SEND (data-vector-id, '_getPrinting', printing);

Argument Type Description
printing C returns the printing status:

'Y' the viewer is printing the data
'N' the viewer is not printing the data

Details
The _getPrinting method is valid only if the viewer is a table editor. Note that the
viewer, not the model, sets the printing status. The model queries the status but does
not set it.

A printing state set to 'Y' informs the model that the viewer is printing its data.
Similar to the measuring status controlled by the _setMeasuring and _getMeasuring
methods, the models are not usually concerned with this unless the model has an
unspecified number of rows or columns. If this is the case, the model needs to check
this value when returning information to the viewer in the _getRowInfo or
_getColumnInfo method call so that it does not continue to return rows or columns
while printing is occurring.

_getProtect

Returns the protect status

Syntax
CALL SEND (data-vector-id, '_getProtect', protect);

Argument Type Description
protect C returns the protect status:

'Y' protected
'N' not protected

Details
The _getProtect method is valid if the viewer is either a form editor or a table editor.
**_getRepresentation**

Returns the representation

---

**Syntax**

CALL SEND (data-vector-id, '_getRepresentation', rep);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rep</td>
<td>C</td>
<td>returns the label representation:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'REGTITLE' label is in the region title</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'NOLABEL' no label is displayed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'WIDGET' label is a widget</td>
</tr>
</tbody>
</table>

**Details**

The _getRepresentation method is valid only if the viewer is a form editor.

---

**_getResizing**

Returns the resizing status

---

**Syntax**

CALL SEND (data-vector-id, '_getResizing', resizing);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resizing</td>
<td>C</td>
<td>returns the resizing status:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' the viewer is setting the column or row width or height</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' the viewer is not setting the column or row width or height</td>
</tr>
</tbody>
</table>

**Details**

The _getResizing method is valid only if the viewer is a table editor. Note that the viewer, not the model, sets the resizing status. The model queries the status but does not set it.

The viewer sets the resizing status to inform the model that the width or height of a column or row is being set by the viewer. This action is known as a software resize. A
software resize can occur due to either a model "fit" request or by the redistribution of space that occurs with nested rows/columns. For example, a software resize occurs when a parent column is wider than its children columns.

The resizing value should be checked by the model in the _setColumnInfo method or the _setRowInfo method. If set, the model should store the label dimension and alignment values (via the _getDimension and _getAlignment methods) so that these values can be returned to the viewer later through the _getColumnInfo or _getRowInfo method.

When the label dimension and alignment values are returned to the viewer, set the alignment via the _setAlignment method. Do not set fixed on _setDimension. Use fixed only for user resizes.

The mode should discard the saved dimension and alignment values on a _clearSoftResizes method. For more information, see the Form Data Model class.

__getReverse__

Returns the reverse status

---

**Syntax**

CALL SEND (data-vector-id, '_getReverse', reverse);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reverse</td>
<td>C</td>
<td>returns the reverse status:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' foreground and background colors are swapped</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' foreground and background colors are not swapped</td>
</tr>
</tbody>
</table>

**Details**

The _getReverse method is valid only if the viewer is a table editor.

__getSoftEndOfData__

Returns the soft end-of-data status

---

**Syntax**

CALL SEND (data-vector-id, '_getSoftEndOfData', soft-eod);
### soft-eod

**C** returns the soft end-of-data status:

- 'Y' this is the soft end of data
- 'N' this is not the soft end of data

**Details**

The _getSoftEndOfData method is valid only if the viewer is a table editor. The soft end of data is related to the hard end of data controlled by the _getEndOfData and _setEndOfData methods.

Both hard and soft end-of-data status are used to notify the viewer that the row or column being requested is past the number of rows or columns that the model has. For example, if there are three columns in a model, on a viewer’s call to _getColumnInfo requesting information about column 4, the model sets the end-of-data status to 'Y'.

The difference between setting the hard end of data and the soft end of data is that with the hard end of data you cannot scroll in the table editor, while with the soft end of data you can scroll. The hard end of data is useful for models with a stable and known number of rows or columns, while the soft end of data is useful for models with a changing or unknown number of rows or columns.

---

### _getText

**Returns the displayed row or column name**

---

**Syntax**

CALL SEND (data-vector-id, '_getText', text);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>C</td>
<td>returns the displayed row or column name</td>
</tr>
</tbody>
</table>

**Details**

The _getText method is valid only if the viewer is a table editor. Use the _getName method if the viewer is a form editor.

---

### _getType

**Returns the column type**

---

**Syntax**

CALL SEND (data-vector-id, '_getType', type);
Row/Column Data Vector Class

_getWrapping

Argument | Type | Description
---------|------|------------------
type     | C | N
returns the column type:
‘C’ character
‘N’ numeric

Details
The _getType method is valid only if the viewer is a form editor. You can use the _getType method only to pass a row/column data vector to the model’s _getColumnInfo or _setColumnInfo method. Its values are not meaningful for row settings.

_getVjust

Returns the vertical justification

Syntax
CALL SEND (data-vector-id, '_getVjust', just);

Argument | Type | Description
---------|------|------------------
just     | C
returns the vertical justification:
‘TOP’
‘MIDDLE’
‘BOTTOM’

Details
The _getVjust method is valid only if the viewer is a table editor.

_getWrapping

Gets the text wrapping information

Syntax
CALL SEND (data-vector-id, '_getWrapping', wrapping<, split-char>);
### Argument Type Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wrapping</td>
<td>C</td>
<td>returns the wrapping status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' the text will be wrapped</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' the text will not be wrapped</td>
</tr>
<tr>
<td>split-char</td>
<td>C</td>
<td>character on which to split lines</td>
</tr>
</tbody>
</table>

### Details

The `getWrapping` method is valid only if the viewer is a table editor.

If wrapping is set to 'Y', the character set by `split-char` is used to wrap the text over multiple lines. For example, if `split-char` is set to the backslash (\) character, the text `One\ Two\ Three` is split over three lines.

---

### _init

**Initialize the data vector**

---

### Syntax

CALL SEND (data-vector-id, '_init'<, data-vector-info>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data-vector-info</td>
<td>N</td>
<td>specifies the identifier of an SCL list that contains initialization information that the data vector can use when initializing</td>
</tr>
</tbody>
</table>

### Details

The `_init` method does the setup of the row/column data vector. It also calls the super `_init` method. The `_init` method is called whenever you make an instance of the Row/Column Data Vector class, either through a call to `_new` or to the instance function.

The data-vector-info list is optional. The list can tell the data vector which type of viewer this data vector will be used with if that information is known. This list can contain named items of NOFORME or NOTABLEE.

If you do not use this list, resources for both the form-specific and table-specific parts of the data vector are allocated, and you are allowed to use data vector methods that deal with information that either viewer will use.

When you create the row/column data vector, if you know that it will only be used in conjunction with the table editor, you can save memory and receive additional error checking by setting a named item of NOFORME to 1 in this list. In that case, only the table-specific resources are allocated, and when you attempt to call methods of the row/column data vector that deal with form-specific information, you receive a program halt.

On the other hand, when you create a row/column data vector, if you know it will only be used in conjunction with the form editor, you can receive additional error checking by setting a named item of NOTABLEE to 1 in this list. In that case, there
are not any resource savings, but when you attempt to call methods of a row/column data vector that deal with table-specific information, you receive a program halt.

The table editor sets the NOFORME item to 1 in the data-vector-info list for the data vectors it uses internally for the resource savings. The form editor does not pass a data-vector-info list for the data vectors it uses internally.

Example

The following example makes a row/column data vector that is used only with the table editor:

Example Code 87.1 Make a row/column data vector for table only.

```millenium
arglist = makelist();
rc = setnitemn(arglist, 1, 'NOFORME');
classid = loadclass('sashelp.fsp.rcdvec');
call send(classid, '_new', rcdvecid,
         arglist);
```

Example Code 87.2 Try to call a form-only method.

```millenium
call send(rcdvecid, '_setClassName',
         'sashelp.fsp.radiobox.class');
```

The above code causes a program halt because it attempts to call a method that is only for the table even though we have specified that this row/column data vector is used only with a form:

ERROR: Table-specific method called on form-specific data vector.
Arguments passed to SEND:
  1 RCDVECID = 33
  2 (Character Literal) = '_setClassName'
Parameters passed to SEND ENTRY:
  1 (Character Literal) = 'sashelp.fsp.radiobox.class'

_setAlignment

Sets the base line or alignment

Syntax

CALL SEND (data-vector-id, '_setAlignment', align<, unit>);
### Argument Type Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>align</td>
<td>N</td>
<td>specifies the base line or alignment value</td>
</tr>
<tr>
<td>unit</td>
<td>C</td>
<td>specifies the base line or alignment unit of measure.</td>
</tr>
</tbody>
</table>

### Details

The _setAlignment method is valid only if the viewer is a table editor. If you use the row/column data vector in conjunction with the _getRowInfo or _setRowInfo method, the value specified by the _setAlignment method is interpreted as the row's base line. This is the distance from the top of the row to the start of the text.

If you use the row/column data vector in conjunction with the _getColumnInfo or _setColumnInfo method, the value returned by the _setAlignment method is interpreted as the row's alignment. This is the distance from the left of the column to the start of the text.

The following conditions also are true when a row/column data vector is used in conjunction with the _getColumnInfo method or the _setColumnInfo method:

- The alignment value specified by the _setAlignment method specifies how much space is available in which to align the text from the left of the column.
- The alignment justification specified by the _setAlignJust method tells how to justify the text horizontally in the width given by the _setAlignment method.
- The alignment character specified by the _setAlignmentChar method is the character on which you want the data in the column to be aligned.
- The number of decimals specified by the _setDecimals method is the number of characters allowed to the right of the alignment character specified by the _setAlignmentChar method if the alignment justification specified by the _setAlignJust method is RIGHT.
- Use either the _setDecimals or the _setAlignment method, but not both.

---

### _setAlignmentChar

Sets the alignment character

#### Syntax

CALL SEND (data-vector-id, '_setAlignmentChar', align-char);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>align-char</td>
<td>C</td>
<td>specifies the alignment character</td>
</tr>
</tbody>
</table>

#### Details

The _setAlignmentChar method is valid only if the viewer is a table editor. You can use the _setAlignmentChar method only to pass a row/column data vector to the model's _getColumnInfo or _setColumnInfo method. Its values are not meaningful for row settings.
The alignment character, when specified, is typically a decimal point, '.', which is usually used to line up numeric values.

The following conditions also are true when you use a row/column data vector in conjunction with the _getColumnInfo method or the _setColumnInfo method:

- The alignment value specified by the _setAlignment method specifies how much space is available in which to align the text from the left of the column.
- The alignment justification specified by the _setAlignJust method tells how to justify the text horizontally in the width given by the _setAlignment method.
- The alignment character specified by the _setAlignmentChar method is the character on which you want the data in the column to be aligned.
- The number of decimals specified by the _setDecimals method is the number of characters allowed to the right of the alignment character specified by the _setAlignmentChar method if the alignment justification specified by the _setAlignJust method is RIGHT.
- Use either the _setDecimals or the _setAlignment method, but not both.

### _setAlignJust

Sets the alignment justification

---

**Syntax**

CALL SEND (data-vector-id, '_setAlignJust', just);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| just     | C    | specifies the alignment justification:  
          |       | 'RIGHT'    |
          |       | 'LEFT'     |
          |       | 'CENTER'   |

**Details**

The _setAlignJust method is valid only if the viewer is a table editor. Use the _setAlignJust method only to pass a row/column data vector instance to the model’s _getColumnInfo or _setColumnInfo method. Its values are not meaningful for row settings.

The following conditions also are true when you use a row/column data vector in conjunction with the _getColumnInfo method or the _setColumnInfo method:

- The alignment value specified by the _setAlignment method specifies how much space is available in which to align the text from the left of the column.
- The alignment justification specified by the _setAlignJust method tells how to justify the text horizontally in the width given by the _setAlignment method.
- The alignment character specified by the _setAlignmentChar method is the character on which you want the data in the column to be aligned.
The number of decimals specified by the _setDecimals method is the number of characters allowed to the right of the alignment character specified by the _setAlignmentChar method if the alignment justification specified by the _setAlignJust method is RIGHT.

Use either the _setDecimals or the _setAlignment method, but not both.

---

### _setBackgroundColor

Sets the background color

**Syntax**

CALL SEND (data-vector-id, '_setBackgroundColor', color);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>C</td>
<td>specifies the background color</td>
</tr>
</tbody>
</table>

**Details**

The _setBackgroundColor method is valid if the viewer is either a form editor or a table editor.

---

### _setBackgroundColorPattern

Sets the background pattern

**Syntax**

CALL SEND (data-vector-id, '_setBackgroundColorPattern', 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 background pattern: '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%' 'TRANSPARENT'</td>
</tr>
</tbody>
</table>

**Details**

The _setBackgroundColorPattern method is valid only if the viewer is a table editor.
_setBorderColor

Sets the border outline color

Syntax

CALL SEND (data-vector-id, '._setBorderColor', area, color);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>area</td>
<td>C</td>
<td>specifies the area to set the border color for:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'ALL' all four sides of the cell</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'RIGHT' the right side of the cell</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'LEFT' the left side of the cell</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'TOP' the top side of the cell</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'BOTTOM' the bottom side of the cell</td>
</tr>
<tr>
<td>color</td>
<td>C</td>
<td>specifies the border outline color</td>
</tr>
</tbody>
</table>

Details

The _setBorderColor method is valid only if the viewer is a table editor. Note that the default label outline style used by the table editor is button outline style. The style and color set by the _setBorderColor and _setBorderStyle methods are not used for button outline style. To change the table editor’s border outline style to no longer use buttons, call the _setAttributes table editor method with BUTTON_LABELS set to ‘N’.

See also the Table Editor class.

_setBorderStyle

Sets the border style

Syntax

CALL SEND (data-vector-id, '._setBorderStyle', area, style);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>area</td>
<td>C</td>
<td>specifies the area to set the border style for:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'ALL' all four sides of the cell</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'RIGHT' the right side of the cell</td>
</tr>
</tbody>
</table>
The _setBorderStyle method is valid only if the viewer is a table editor. Note that the default label outline style used by the table editor is button outline style. The style and color set by the _setBorderColor and _setBorderStyle methods are not used for button outline style. To change the table editor's border outline style to no longer use buttons, call the _setAttributes table editor method with BUTTON_LABELS set to 'N'.

See also the Table Editor class.

*_setBorderWidth*

Sets the border width

**Syntax**

CALL SEND (data-vector-id, '_setBorderWidth', area, width, unit);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>area</td>
<td>C</td>
<td>specifies the area to set the border width for:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'ALL' all four sides of the cell</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'RIGHT' the right side of the cell</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'LEFT' the left side of the cell</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'TOP' the top side of the cell</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'BOTTOM' the bottom side of the cell</td>
</tr>
<tr>
<td>width</td>
<td>N</td>
<td>specifies the border width</td>
</tr>
<tr>
<td>unit</td>
<td>C</td>
<td>specifies the border width unit of measure.</td>
</tr>
</tbody>
</table>

**Details**

The _setBorderWidth method is valid only if the viewer is a table editor.
_setClassName

Sets the label class name

Syntax

CALL SEND (data-vector-id, '_setClassName', class);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>C</td>
<td>specifies the label class name</td>
</tr>
</tbody>
</table>

Details

The _setClassName method is valid only if the viewer is a form editor. The label class is the name of the widget class the form editor uses to display labels. The instance variables list of this class is controlled with the _getAttributes and _setAttributes methods.

_setClassProperties

Sets the label attributes list

Syntax

CALL SEND (data-vector-id, '_setClassProperties', attributes);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes</td>
<td>N</td>
<td>specifies the identifier of an SCL list that contains the label attributes</td>
</tr>
</tbody>
</table>

Details

The _setClassProperties method is valid only if the viewer is a form editor. The attributes list is the instance variable list of the label class used by the form editor. For example, if the form editor is displaying its labels with the Text Entry class, the attributes list reflects the instance variables list of the Text Entry class. The class being used by the form editor is controlled with the _getClassNamed and _setClassName methods.
_setColor

Sets the foreground color

Syntax
CALL SEND (data-vector-id, '_setColor', color);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>C</td>
<td>specifies the foreground color</td>
</tr>
</tbody>
</table>

Details
The _setColor method is valid if the viewer is either a form editor or a table editor.

_setCoordinates

Sets the row or column coordinates

Syntax
CALL SEND (data-vector-id, '_setCoordinates', cur-label);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cur-label</td>
<td>N</td>
<td>specifies the identifier of an SCL list that contains the row or column coordinates.</td>
</tr>
</tbody>
</table>

Details
The _setCoordinates method is valid if the viewer is either a form editor or a table editor.

The current row or column indicates to which row or column this row/column data vector refers. Each row/column data vector deals only with one row or column at a time. If this row/column data vector is used as an argument to the _getRowInfo or _setRowInfo method, this value is interpreted as the current row. If this Row/Column Data Vector instance is used as an argument to the _getColumnInfo or _setColumnInfo method, this value is interpreted as the current column.

_setDecimals

Sets the number of decimal places
Syntax

CALL SEND (data-vector-id, '_setDecimals', decimals);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>decimals</td>
<td>N</td>
<td>specifies the number of decimal places used for alignment</td>
</tr>
</tbody>
</table>

Details

The _setDecimals method is valid only if the viewer is a table editor. You can use the _setDecimals method only to pass a row/column data vector to the model's _getColumnInfo or _setColumnInfo method. Its values are not meaningful for row settings.

The following conditions also are true when you use a row/column data vector in conjunction with the _getColumnInfo method or the _setColumnInfo method:

- The alignment value specified by the _setAlignment method specifies how much space is available in which to align the text from the left of the column.
- The alignment justification specified by the _setAlignJust method tells how to justify the text horizontally in the width given by the _setAlignment method.
- The alignment character specified by the _setAlignmentChar method is the character on which you want the data in the column to be aligned.
- The number of decimals specified by the _setDecimals method is the number of characters allowed to the right of the alignment character specified by the _setAlignmentChar method if the alignment justification specified by the _setAlignJust method is RIGHT.
- Use either the _setDecimals or the _setAlignment method, but not both.

The decimals value is used only if there is an alignment character set.

_setDimension

Sets the width or height

Syntax

CALL SEND (data-vector-id, '_setDimension', dim<, unit<, add<, fit<, fixed>>>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dim</td>
<td>N</td>
<td>specifies the width or height value</td>
</tr>
<tr>
<td>unit</td>
<td>C</td>
<td>specifies the width or height unit of measure.</td>
</tr>
<tr>
<td>add</td>
<td>C</td>
<td>specifies whether the width or height is added to the margins: 'Y' add</td>
</tr>
</tbody>
</table>
### _setDimension

**Argument Type Description**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fit</td>
<td>C</td>
<td>specifies whether the width or height is a minimum size:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' minimum size</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' not a minimum size</td>
</tr>
<tr>
<td>fixed</td>
<td>C</td>
<td>specifies whether the width or height is fixed by the viewer:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' fixed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' not fixed</td>
</tr>
</tbody>
</table>

#### Details

The _setDimension method is valid only if the viewer is a table editor.

If fit is set to 'Y', the viewer interprets the width or height as a minimum size. This means that the viewer attempts to measure the data to determine their size, but it never goes below the width or height specified.

If fixed is set to 'Y', the viewer does not attempt to adjust the size of the data. Margins are ignored, and the viewer does not attempt to ensure that the row or column is big enough to display all of the label or the data.

---

### _setEndOfData

**Sets the hard end-of-data status**

**Syntax**

CALL SEND (data-vector-id, '_setEndOfData', end);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>end</td>
<td>C</td>
<td>specifies the hard end-of-data status:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' this is the hard end of data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' this is not the hard end of data</td>
</tr>
</tbody>
</table>

#### Details

The _setEndOfData method is valid if the viewer is either a form editor or a table editor. The hard end of data is related to the soft end of data controlled by the _getSoftEndOfData and _setSoftEndOfData methods.

Both hard and soft end-of-data status are used to notify the viewer that the row or column being requested is past the number of rows or columns that the model has. For example, if there are three columns in a model, on a viewer’s call to _getColumnInfo requesting information about column 4, the model sets the end-of-data status to 'Y'.

The difference between setting the hard end of data and the soft end of data is that with the hard end of data you cannot scroll in the table editor, while with the soft end
of data you can scroll. The hard end of data is useful for models with a stable and known number of rows or columns, while the soft end of data is useful for models with a changing or unknown number of rows or columns.

_setFont

Sets the font

Syntax

CALL SEND (data-vector-id, '_setFont', font-list-id);

<table>
<thead>
<tr>
<th>Argument</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 information</td>
</tr>
</tbody>
</table>

Details

The _setFont method is valid if the viewer is either a form editor or a table editor. The list sent to the _setFont method should come from a method call to retrieve the font from any other object that saves its font in a list format or from a call to the SCL FONTSEL function. For example, you can call _getFont on one row/column data vector to save the font and then call _setFont to set that font on another row/column data vector.

_setFormat

Sets the format

Syntax

CALL SEND (data-vector-id, '_setFormat', format);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>C</td>
<td>specifies the format</td>
</tr>
</tbody>
</table>

Details

The _setFormat method is valid only if the viewer is a form editor. The format must be a valid SAS format for the type.
_setFormatLength

Sets the format length

Syntax
CALL SEND (data-vector-id, '_setFormatLength', length);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>N</td>
<td>specifies the format length</td>
</tr>
</tbody>
</table>

Details
The _setFormatLength method is valid if the viewer is either a form editor or a table editor.
The value returned by the _setFormatLength method is the maximum number of characters the viewer will display.

_setHidden

Sets the hidden status

Syntax
CALL SEND (data-vector-id, '_setHidden', hidden);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hidden</td>
<td>C</td>
<td>specifies the hidden status:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' hidden</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' not hidden</td>
</tr>
</tbody>
</table>

Details
The _setHidden method is valid only if the viewer is a form editor.

_setHjust

Sets the horizontal justification
Syntax
CALL SEND (data-vector-id, '_setHjust', just);

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

Details
The _setHjust method is valid if the viewer is either a form editor or a table editor.

_setInformat

Sets the informat

Syntax
CALL SEND (data-vector-id, '_setInformat', informat);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>informat</td>
<td>C</td>
<td>specifies the informat</td>
</tr>
</tbody>
</table>

Details
The _setInformat method is valid only if the viewer is a form editor. The informat must be a valid SAS informat for the type.

_setInherit

Sets the inherit status

Syntax
CALL SEND (data-vector-id, '_setInherit', inherit);
Argument | Type | Description
---|---|---
inherit | C | specifies the inherit status:
| | Y | the label class inherits its attributes from the label class rather than from the settings in the form editor attributes screens
| | N | the label class uses the fonts, colors, and other attributes from the form editor attributes screens

Details
The _setInherit method is valid only if the viewer is a form editor. The default inherit status is 'N'. If inherit is set to 'Y', the colors, fonts, and other attributes specified under the form editor attributes screens are ignored in favor of the attributes set in the specified label class.

_setLightSource
Sets the light source for button borders

Syntax
CALL SEND (data-vector-id, '_setLightSource', light-source);

Where ...
---
light-source | C | specifies the light source for button borders:
| | LOWER LEFT'
| | LOWER RIGHT'
| | UPPER LEFT'
| | UPPER RIGHT'

Details
The _setLightSource method is valid only if the viewer is a table editor.

_setMargin
Sets the margin settings

Syntax
CALL SEND (data-vector-id, '_setMargin', area, margin, unit);
Row/ Column Data Vector Class _ setMeasuring

## Argument Type Description

**area**  
C specifies the area to set the margin setting for:

- ‘ALL’ all four sides of the cell
- ‘RIGHT’ the right side of the cell
- ‘LEFT’ the left side of the cell
- ‘TOP’ the top side of the cell
- ‘BOTTOM’ the bottom side of the cell

**margin**  
N specifies the margin setting value

**unit**  
C specifies the margin setting unit of measure.

### Details
The _setMargin method is valid only if the viewer is a table editor.

### _setMeasuring
Sets the measuring status

#### Syntax

CALL SEND (data-vector-id, 'setMeasuring', measuring);

**Argument Type Description**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>measuring</td>
<td>C</td>
<td>specifies the measuring status:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' the viewer is measuring the data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' the viewer is not measuring the data</td>
</tr>
</tbody>
</table>

### Details
The _setMeasuring method is valid only if the viewer is a table editor.

Note that the viewer, not the model, sets the measuring status. The model queries the status but does not set it.

A measuring state set to ‘Y’ informs the model that the viewer is measuring its data to determine how large to make its cells. Similar to the printing status controlled by the _setPrinting and _getPrinting methods, the models are not usually concerned with this unless the model has an unspecified number of rows or columns. If this is the case, the model needs to check this value when returning information to the viewer in the _getRowInfo or _getColumnInfo method call so that it does not continue to return rows or columns while measuring is occurring.
_setNameJust

Sets the number of columns for justification

Syntax
CALL SEND (data-vector-id, 'setNameJust', num-columns);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num-columns</td>
<td>N</td>
<td>specifies the number of columns under which the column name should be justified:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 default (use one column)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-1 use the whole table</td>
</tr>
</tbody>
</table>

Details
The _setNameJust method is valid only if the viewer is a table editor. You can use the _setNameJust method only to pass a row/column data vector to the model’s _getRowInfo or _setRowInfo method. Its values are not meaningful for column settings.

The default value of 1 for num-columns indicates to justify the column label under one column. A value of -1 for num-columns indicates to justify the column label under the entire table. This has the effect of making that column label span the entire table. Values other than 1 and -1 are not supported.

_setPrinting

Sets the printing status

Syntax
CALL SEND (data-vector-id, 'setPrinting', printing);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printing</td>
<td>C</td>
<td>specifies the printing status:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' the viewer is printing the data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' the viewer is not printing the data</td>
</tr>
</tbody>
</table>

Details
The _setPrinting method is valid only if the viewer is a table editor. Note that the viewer, not the model, sets the printing status. The model queries the status but does not set it.
A printing state set to 'Y' informs the model that the viewer is printing its data. Similar to the measuring status controlled by the _setMeasuring and _getMeasuring methods, the models are not usually concerned with this unless the model has an unspecified number of rows or columns. If this is the case, the model needs to check this value when returning information to the viewer in the _getRowInfo or _getColumnInfo method call so that it does not continue to return rows or columns while printing is occurring.

__setProtect

Sets the protect status

**Syntax**

CALL SEND (data-vector-id, '_setProtect', protect);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>protect</td>
<td>C</td>
<td>specifies the protect status:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' protected</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' not protected</td>
</tr>
</tbody>
</table>

**Details**

The _setProtect method is valid if the viewer is either a form editor or a table editor.

__setRepresentation

Sets the label representation

**Syntax**

CALL SEND (data-vector-id, '_setRepresentation', rep);
Chapter 87

### Argument Type Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rep</td>
<td>C</td>
<td>specifies the label representation:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'REGTITLE' label is in the region title</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'NOLABEL' no label is displayed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'WIDGET' label is a widget</td>
</tr>
</tbody>
</table>

**Details**

The _setRepresentation method is valid only if the viewer is a form editor.

---

### _setResizing

Sets the resizing status

---

### Syntax

CALL SEND (data-vector-id, '_setResizing', resizing);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resizing</td>
<td>C</td>
<td>specifies the resizing status:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' the viewer is setting the column or row width or height</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' the viewer is not setting the column or row width or height</td>
</tr>
</tbody>
</table>

**Details**

The _setResizing method is valid only if the viewer is a table editor. Note that the viewer, not the model, sets the resizing status. The model queries the status but does not set it.

The viewer sets the resizing status to inform the model that the width or height of a column or row is being set by the viewer. This action is known as a software resize. A software resize can occur due to either a model “fit” request or by the redistribution of space that occurs with nested rows/columns. For example, a software resize occurs when a parent column is wider than its children columns.

The resizing value should be checked by the model in the _setColumnInfo method or the _setRowInfo method. If set, the model should store the label dimension and alignment values (via the _getDimension and _getAlignment methods) so that these values can be returned to the viewer later through the _getColumnInfo or _getRowInfo method.

When the label dimension and alignment values are returned to the viewer, set the alignment via the _setAlignment method. Do not set fixed on _setDimension. Use fixed only for user resizes.

The model should discard the saved dimension and alignment values on a _clearSoftResizes method. For more information, see the Form Data Model class.
**_setReverse_**

Sets the reverse status

---

**Syntax**

CALL SEND (data-vector-id, '_setReverse', reverse);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reverse</td>
<td>C</td>
<td>specifies the reverse status:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' swap foreground and background colors</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' do not swap foreground and background colors</td>
</tr>
</tbody>
</table>

**Details**

The _setReverse method is valid only if the viewer is a table editor.

---

**_setSoftEndOfData_**

Sets the soft end-of-data status

---

**Syntax**

CALL SEND (data-vector-id, '_setSoftEndOfData', soft-eod);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>soft-eod</td>
<td>C</td>
<td>specifies the soft end-of-data status:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' this is the soft end of data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' this is not the soft end of data</td>
</tr>
</tbody>
</table>

**Details**

The _setSoftEndOfData method is valid only if the viewer is a table editor. The soft end of data is related to the hard end of data controlled by the _getEndOfData and _setEndOfData methods.

Both hard and soft end-of-data status are used to notify the viewer that the row or column being requested is past the number of rows or columns that the model has. For example, if there are three columns in a model, on a viewer’s call to _getColumnInfo requesting information about column 4, the model sets the end-of-data status to 'Y'.

The difference between setting the hard end of data and the soft end of data is that with the hard end of data you cannot scroll in the table editor, while with the soft end
of data you can scroll. The hard end of data is useful for models with a stable and known number of rows or columns, while the soft end of data is useful for models with a changing or unknown number of rows or columns.

_setText

**Sets the displayed row or column name**

**Syntax**

CALL SEND (data-vector-id, '_setText', text);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>C</td>
<td>specifies the displayed row or column name</td>
</tr>
</tbody>
</table>

**Details**

The _setText method is valid only if the viewer is a table editor.

_setVjust

**Sets the vertical justification**

**Syntax**

CALL SEND (data-vector-id, '_setVjust', just);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>just</td>
<td>C</td>
<td>specifies the vertical justification:</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 _setVjust method is valid only if the viewer is a table editor.
_setWrapping

Sets the text wrapping information

Syntax
CALL SEND (data-vector-id, '_setWrapping', wrapping<, split-char>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wrapping</td>
<td>C</td>
<td>specifies the wrapping status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' the text will be wrapped</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' the text will not be wrapped</td>
</tr>
<tr>
<td>split-char</td>
<td>C</td>
<td>character on which to split lines</td>
</tr>
</tbody>
</table>

Details
The _setWrapping method is valid only if the viewer is a table editor.

If wrapping is set to 'Y', the character set by split-char is used to wrap the text over multiple lines. For example, if split-char is set to the backslash (\) character, the text One\Two\Three is split over three lines.