Overview

The GET_DATA Data Vector 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.

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

Most users of the Data Table and Data Form classes need not be concerned with the GET_DATA Data Vector class. In the Data Table and Data Form classes, the GET_DATA Data Vector class is used internally.

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

- call certain methods of your own instance of the model class directly
- 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 GET_DATA Data Vector class are

- _getData
- _setData

The _getData and _setData methods are methods of the Form Data Model class (see the Form Data Model class). The model subclasses of the Form 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 the Form Data
Model class, the GET_DATA Data Vector class is used by all models’ _getData and _setData methods.

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

The GET_DATA 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 Model class uses an instance of the GET_DATA Data Vector class in its _getData method to fill in values for the viewer and in its _setData method to set values for the viewer.

The _getData method of the model takes an instance of a GET_DATA Data Vector class as an argument. When the model is called, the viewer is responsible for creating that instance of the GET_DATA Data Vector class. The model’s _getData method then fills in the elements of the GET_DATA Data Vector class passed to it. When control returns to the viewer, the viewer queries the elements set in the instance passed to the model.

Similarly, the _setData method of the model takes an instance of the GET_DATA Data Vector class as an argument. When the model is called with the _setData method, the viewer is responsible for creating that instance of the GET_DATA Data Vector class. In the _setData method call, the viewer is responsible for filling in the elements of the GET_DATA data vector that it expects to be set in the model. The model then queries the elements set in the GET_DATA data vector it receives and sets values accordingly.

Parent:
- sashelp.fsp.object.class

Class:
- Sashelp.fsp.GDDVec.class

---

Using the GET_DATA Data Vector Class

Calling Methods of the GET_DATA Data Vector Class

**CAUTION:**

Follow these Instructions Carefully To use the GET_DATA Data Vector class correctly, you must follow the instructions in this section.

The GET_DATA Data Vector class contains information about one or more columns. The GET_DATA Data Vector class contains such information because the viewer can request or set information about as many columns as it likes with one call to the model’s _getData or _setData method. The GET_DATA Data Vector class, however, contains information about only one row because the model’s _getData and _setData methods only operate with one row at a time.

The caller of the model’s _getData or _setData method is responsible for creating the instance of the GET_DATA Data Vector class passed as an argument to these methods. Therefore, if you override the model’s _getData or _setData method, you need not create your own instance of the GET_DATA Data Vector class; one will be passed to you. If,
however, you call the _getData or _setData method of the model directly or if your own
viewer class calls _getData or _setData on the model, you are responsible for creating
your own instance of the GET_DATA Data Vector class.
If you make your own instance of a GET_DATA Data Vector class, you must set up
the GET_DATA data vector with information about
  □ how many columns exist
  □ to which column the GET_DATA Data Vector method calls refer.

After making an instance of the GET_DATA Data Vector class, you should first call
the _setNumberOfColumns method to tell the GET_DATA data vector how many
columns exist. Then call _setRow to tell the GET_DATA data vector which row the
instance refers to.

Next, for each column in the instance, you should call the _setIndex method, followed
by the _setColumn method, to tell the GET_DATA data vector the information about
which column you are requesting information for in each of the columns declared with
the _setNumberOfColumns method.

Then you should call the _setIndex method to declare which column further method
calls will refer to. If at a later point you want to change which column method calls
refer to, call the _setIndex method again with the new column number. All methods of
the GET_DATA Data Vector class other than the _setNumberOfColumns and _setIndex
method apply to the current column, that is, the column set by the last _setIndex
method call.
Calling the _setColumn method on each column index is necessary because the
columns in the GET_DATA data vector may be noncontiguous. For example, if you are
using a table editor, it may call _getData on the model with an instance of the
GET_DATA Data Vector class requesting information about two columns. But the first
column the table editor wants information about may be column number 6, and the
second column it wants information about may be column number 3. The current
column set by the _setColumn method reflects the display order that the table editor is
using, which is 6 followed by 3 in this case.

Example: Calling GET_DATA Data Vector Methods

The following example is a code fragment from a situation in which a developer has
created a viewer. In this code fragment, a GET_DATA data vector is created and set up
with initial values. After that, the GET_DATA data vector can be sent to a model on a
_setData call.
If this code fragment were used as part of an application, any objects that were
instantiated during execution would need to be terminated when those objects were no
longer needed. Likewise, any lists that were created during execution would need to be
deleted when those lists were no longer needed.

Make an instance of the GET_DATA Data Vector class.

```plaintext
    gddvecid = instance(loadclass('sashelp.fsp.
                 gddvec'));
```

Declare the instance to contain information about two columns.

```plaintext
    call send(gddvecid, '_setNumberOfColumns',
                2);
```
This data vector deals with row number 6.

\begin{verbatim}
   rowlst = makelist();
   rc = insertn(rowlst, 6, -1);
   call send(gddvecid, '_setRow', rowlst);
\end{verbatim}

Set the first column’s color to green. The first column to be set up is column number 4.

\begin{verbatim}
   call send(gddvecid, '_setIndex', 1);
   collst1 = makelist();
   rc = insertn(collst1, 4, -1);
   call send(gddvecid, '_setColumn', collst1);
   call send(gddvecid, '_setColor', 'green');
\end{verbatim}

Set the second column to use center horizontal justification. The second column to be set up is column number 5.

\begin{verbatim}
   call send(gddvecid, '_setIndex', 2);
   collst2 = makelist();
   rc = insertn(collst2, 5, -1);
   call send(gddvecid, '_setColumn', collst2);
   call send(gddvecid, '_setHjust', 'center');
\end{verbatim}

Some of the GET_DATA 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 GET_DATA 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 GET_DATA Data Vector class.

However, if you know when you make a GET_DATA data vector that it will only be used for either a form editor or a table editor, then the GET_DATA 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 GET_DATA Data Vector class can be used with the Form Editor class and which can be used with the Table Editor class.

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Used in Form Editor?</th>
<th>Used in Table Editor?</th>
</tr>
</thead>
<tbody>
<tr>
<td>_getBackgroundColor</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>_getBackgroundPattern</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>_getBorderColor</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>_getBorderStyle</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>_getBorderWidth</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 70.1** Where Methods Can Be Used
<table>
<thead>
<tr>
<th>Method Name</th>
<th>Used in Form Editor?</th>
<th>Used in Table Editor?</th>
</tr>
</thead>
<tbody>
<tr>
<td>_getClassName</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>_getClassProperties</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>_getColor</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>_getColumn</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>_setFont</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>_getFormatLength</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>_getHjust</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>_getInherit</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>_getInternalERROR</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>_getLightSource</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>_getMargin</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>_getMeasuring</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>_getNumberOfColumns</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>_getPrinting</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>_getProtect</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>_getReverse</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>_getRow</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>_getText</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>_getType</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>_getUnformattedText</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>_getUnformattedValue</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>_getUserERROR</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>_getVjust</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>_getWrapping</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>_setBackgroundColor</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>_setBackgroundPattern</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>_setBorderColor</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>_setBorderStyle</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>_setBorderWidth</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>_setClassName</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>_setClassProperties</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>_setColor</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>_setColumn</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>_setFont</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>_setFormatLength</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>_setHjust</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
# Using the GET_DATA Data Vector Class with a Form Editor

You need to make special considerations when using the GET_DATA Data Vector class with a form editor. When a form editor displays its data, it calls the _getData method on the first row of data to display, followed by calls to the model _getData method for subsequent rows. For performance reasons, most items in the GET_DATA Data Vector class are only checked for the first row that the viewer calls _getData on. Only the first row is checked because most models have the same settings for a particular column for each row that column is in. For example, in a particular model, the first column may always be red and the second column may always use a particular font. So, when the form editor calls the model with the _getData method for the first displayed row, it saves these color and font values and will not check them again for subsequent _getData calls for other rows. The form editor does check for a few items such as the data value and type, but most other items, such as font, color, and justification, are not checked.

If you are overriding the model's _getData method, consider the following about checking: If you know that your columns will always have the same characteristics regardless of the row, then you need not be concerned. If, however, you want some rows to have different characteristics for the same column, then you need to notify the viewer that this is the case so that it will know to check your settings.

To notify the viewer that it needs to check the settings for all rows, call the viewer's _updateColumn method from within the _getData method call. You should call the _updateColumn method at the end of the _getData model method. You can obtain the viewer's identifier from the model's VIEWERID instance variable.

See the Form Data Model class.

---

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Used in Form Editor?</th>
<th>Used in Table Editor?</th>
</tr>
</thead>
<tbody>
<tr>
<td>_setIndex</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>_setInherit</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>_setInternalERROR</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>_setLightSource</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>_setMargin</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>_setMeasuring</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>_setNumberOfColumns</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>_setPrinting</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>_setProtect</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>_setReverse</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>_setRow</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>_setText</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>_setUnformattedText</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>_setUnformattedValue</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>_setUserERROR</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>_setVjust</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>_setWrapping</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Steps for Overriding the _getData Method with a Form Editor

To subclass Data Set Data Model and display the new subclass in a FRAME entry that contains a form editor object, follow these steps:

1. Subclass the Data Set Data Model class.
   a. Edit `newdatam.class`.
   b. Set the parent to `sashelp.fsp.data_m.class`.
   c. Select methods and override _getData using a label of `getdata`.
   d. Edit the SCL source and copy the contents of `MYDATAM.SCL`.
   e. Compile the SCL source.

2. Create a form editor object.
   a. Edit `new.frame`.
   b. Create an empty region in the FRAME entry.
   c. Use the command `RM FILL 'FORM EDITOR'`.
   d. Set the class to be `newdatam.class`.
   e. Select data attributes and define a data set to use. (The following example uses `sasuser.class`.)
   f. Save and test the FRAME entry.

Example of Overriding the _getData Method with a Form Editor

The following SCL code produces a form editor:

```scl
getdata:
  method gddvecid numcols 8;
  call super(_SELF_, '_getData', gddvecid, numcols);
  call send(gddvecid, '_setIndex', 1);
  rowlst = makelist();
  call send(gddvecid, '_getRow', rowlst);
  rownum = getitemn(rowlst, 1);
  viewerid = getnitemn(_SELF_, 'VIEWERID');
  collst = makelist();
  if rownum eq 1 then do;
    call send(gddvecid, '_setIndex', 1);
    rc = insertn(collst, 1, -1);
    call send(gddvecid, '_setUnformattedText',
      'newtext');
  end;
```

Make row 1, column 1 have new text.

```scl
if rownum eq 1 then do;
  call send(gddvecid, '_setIndex', 1);
  rc = insertn(collst, 1, -1);
  call send(gddvecid, '_setUnformattedText',
    'newtext');
end;
```

Make row 1, column 1 have a smaller font.

```scl
if rownum eq 1 then do;
```
call send(gddvecid, '_setIndex', 1);
rc = insertn(collst, 1, -1);
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(gddvecid, '_setFont', flist);
end;

Make row 1, column 2 use a radio box.

if rownum eq 0 then do;
call send(gddvecid, '_setIndex', 2);
rc = insertn(collst, 2, -1);
call send(gddvecid, '_setClassName',
   'sashelp.fsp.radiobox.class');
oid = loadclass('sashelp.fsp.radiobox');
ivlist = getniteml(oid, '_iv');
ivlist2 = copylist(ivlist, 'Y');
rc = setlattr(ivlist2, 'UPDATE');

Set label length to 6.

rc = setnitemn(ivlist2, 6, 'LENGTH');

Set number of items to 2.

rc = setnitemn(ivlist2, 2, 'NROWS');

Set up values list.

rlist = makelist();
radio1 = makelist();
radio2 = makelist();
rc = setnitemc(radio1, 'Male', 'ROWTXT');
rc = setnitemc(radio1, 'M', 'RETVAL');
rc = setnitemc(radio2, 'Female', 'ROWTXT');
rc = setnitemc(radio2, 'F', 'RETVAL');
rc = setniteml(rlist, radio1, 'R1');
rc = setniteml(rlist, radio2, 'R2');
rc = setniteml(ivlist2, rlist, 'RADIOLST');
rc = putlist(ivlist2);
call send(gddvecid, '_setClassProperties',
        ivlist2);
end;

Make row 1, column 3 use red foreground color.

if rownum eq 1 then do;
    call send(gddvecid, '_setIndex', 3);
    rc = insertn(collst, 3, -1);
    call send(gddvecid, '_setColor', 'RED');
end;

Make row 1, column 4 use green background color.

if rownum eq 1 then do;
    call send(gddvecid, '_setIndex', 4);
    rc = insertn(collst, 4, -1);
    call send(gddvecid, '_setBackgroundColor',
              'GREEN');
end;

Make row 1, column 5 use center justification.

if rownum eq 1 then do;
    call send(gddvecid, '_setIndex', 5);
    rc = insertn(collst, 5, -1);
    call send(gddvecid, '_setHjust', 'CENTER');
end;

call send(viewerid, '_updateColumn', collst);

Delete the lists.

rowlst = dellist (rowlst);
collst = dellist (collst);
flist = dellist (flist);
rlist = dellist (rlist);
radio1 = dellist (radio1);
radio2 = dellist (radio2);
endmethod;
Steps for Overriding the _getData Method with a Table Editor

To subclass Data Set Data Model and display the new subclass in a FRAME entry that contains a table editor object, follow these steps:

1. Subclass the Data Set Data Model class.
   a. Edit `newdatam.class`.
   b. Set parent to `sashelp.fsp.data_m.class`.
   c. Select methods and override _getData using a label of `getdata`.
   d. Edit SCL source and copy the contents of `MYDATAM.SCL`.
   e. Compile the SCL source.

2. Create a table editor object.
   a. Edit `new.frame`.
   b. Create an empty region in the FRAME entry.
   c. Use the command `RM FILL 'TABLE EDITOR'`.
   d. Set the class to be `newdatam.class`.
   e. Select data attributes and define a data set to use. (The following example uses `sasuser.houses`.)
   f. Set the mode to EDIT and the level to MEMBER.
   g. Save and test the FRAME entry.

Example of Overriding the _getData Method with a Table Editor

The following SCL code produces a table editor:

```sas
getdata:
method gddvecid numcols 8;
call super(_SELF_, '_getData', gddvecid,
   numcols);
call send(gddvecid, '_setIndex', 1);
rowlst = makelist();
call send(gddvecid, '_getRow', rowlst);
rownum = getitemn(rowlst, 1);

if rownum eq 1 then do;
call send(gddvecid, '_setIndex', 1);
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');
```

Make row 1, column 1 have a smaller font.
flist = setnitemn(flist, 0.46875, 'RATIO');
call send(gddvecid, '_setFont', flist);
end;

Make row 1, column 3 use red foreground color.

if rownum eq 1 then do;
   call send(gddvecid, '_setIndex', 3);
   call send(gddvecid, '_setColor', 'RED');
end;

Make row 2, column 2 use green background color.

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

Make row 2, column 4 use center justification.

if rownum eq 2 then do;
   call send(gddvecid, '_setIndex', 4);
   call send(gddvecid, '_setHjust', 'CENTER');
end;

Make row 3, column 1 use middle justification.

if rownum eq 3 then do;
   call send(gddvecid, '_setIndex', 1);
   call send(gddvecid, '_setVjust', 'MIDDLE');
end;

Make row 3, column 3 use yellow borders.

if rownum eq 3 then do;
   call send(gddvecid, '_setIndex', 3);
   call send(gddvecid, '_setBorderColor',
               'ALL', 'YELLOW');
   call send(gddvecid, '_setBorderWidth',
               'ALL', 0.1, 'in');
end;
Example of Overriding the _getData Method with a Table Editor

Chapter 70

Make row 4, column 2 use dashed borders.

```markdown
if rownum eq 4 then do;
    call send(gddvecid, '_setIndex', 2);
    call send(gddvecid, '_setBorderStyle', 'ALL', 'DASHED');
    call send(gddvecid, '_setBorderWidth', 'ALL', 0.1, 'in');
end;
```

Make row 4, column 4 use .1 inch borders.

```markdown
if rownum eq 4 then do;
    call send(gddvecid, '_setIndex', 4);
    call send(gddvecid, '_setBorderWidth', 'ALL', 0.1, 'in');
end;
```

Make row 5, column 1 use .3 inch left margin.

```markdown
if rownum eq 5 then do;
    call send(gddvecid, '_setIndex', 1);
    call send(gddvecid, '_setMargin', 'LEFT', 0.3, 'in');
end;
```

Make row 5, column 3 use reverse video.

```markdown
if rownum eq 5 then do;
    call send(gddvecid, '_setIndex', 3);
    call send(gddvecid, '_setReverse', 'Y');
end;
```

Make row 6, column 1 have new text.

```markdown
if rownum eq 6 then do;
    call send(gddvecid, '_setIndex', 1);
    call send(gddvecid, '_setText', 'newtext');
end;
```

Delete the lists.

```markdown
rowlst = dellist (rowlst);
```
Methods

Methods specific to the GET_DATA Data Vector class are described here. Inherited methods are described in the Object class.

Dictionary

_getBackgroundColor

Returns the background color

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| color    | C    | returns the background color.

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

_getBackgroundPattern

Returns the background pattern

Syntax
CALL SEND (data-vector-id, '_getBackgroundPattern', pattern);
### _getBackgroundPattern

<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'</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>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></td>
</tr>
<tr>
<td>style</td>
<td>C</td>
<td>returns the border outline color</td>
</tr>
</tbody>
</table>
### Argument Type Description

<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' 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>style</td>
<td>C</td>
<td>returns the border style</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'SOLID' solid line</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'DOTTED' dotted line</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'DASHED' dashed line</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' 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>returns the border width</td>
</tr>
<tr>
<td>unit</td>
<td>C</td>
<td>returns the unit of measure for the baseline or alignment.</td>
</tr>
</tbody>
</table>

### Details

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

Returns the data class name

---------

Syntax

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 data class name</td>
</tr>
</tbody>
</table>

Details

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

_getClassProperties

Returns the data attributes list

---------

Syntax

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>specifies the identifier of an SCL list to contain the data 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 data class used by the form editor. For example, if the form editor displays its data with the Text Entry class, the attributes list reflects the instance variables list of the Text Entry class. The class used by the form editor is controlled with the _getClassName and _setClassName methods.

_getColor

Returns the foreground color
**_getFont_**

*Returns the font*

**Syntax**

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

---

**_getColumn_**

*Returns the column coordinates*

**Syntax**

CALL SEND (data-vector-id, '_getColumn', column);

---

### Argument Type Description

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

**Details**

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

---

**_getFont_**

*Returns the font*

**Syntax**

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

---

**Syntax**

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

---

**Argument Type Description**

<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 a form editor or a table 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 a form editor or a table editor. The value returned by the _getFormatLength method is the maximum number of characters that the viewer can display.

---

**_setFont_**

*Details*

The _setFont method is valid if the viewer is a form editor or a table editor. You should not directly modify the list returned by the _setFont method, but you can send the list to other objects that take font lists. Changing items in this list and then sending the changed list to the _setFont method can produce unreliable results.

**_getFont_**

*Details*

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

**_getFont_**

*Details*

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

---

**_getHjust_**

*Returns the horizontal justification*

**Syntax**

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

**Details**

The _getHjust method is valid if the viewer is a form editor or a table editor. You should not directly modify the list returned by the _getHjust method, but you can send the list to other objects that take font lists. Changing items in this list and then sending the changed list to the _setFont method can produce unreliable results.
### getHjust

Returns the horizontal justification:

- 'RIGHT'
- 'LEFT'
- 'CENTER'
- 'NONE' (The model does not specify justification; therefore, use the viewer’s default specification.)

**Details**

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

### _getInherit

Returns the inherit status

**Syntax**

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

**Argument**

<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 data class inherits its attributes from the data class rather than from the settings in the form editor attributes screens</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' the data class will use the fonts, colors, and other attributes from the form editor attributes screens</td>
</tr>
</tbody>
</table>

**Details**

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

### _getInternalError

Returns the internal error state
Syntax
CALL SEND (data-vector-id, '_getInternalError', error);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error</td>
<td>C</td>
<td>returns the error state:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' an internal error has occurred in this column</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' an internal error has not occurred in this column</td>
</tr>
</tbody>
</table>

Details
The _getInternalError method is valid only if the viewer is a form editor. The default error state is 'N'. The internal error state is set to 'Y' if an error occurs in the data widget (such as an invalid type) or if a validation error occurs in the model (such as a value that exceeds the maximum). This way of indicating errors differs from a call to _erroronColumn for a column. For more information on those types of errors, see the _getUserError and _setUserError methods.

_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);

<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></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>margin</td>
<td>N</td>
<td>returns the margin setting value</td>
</tr>
<tr>
<td>unit</td>
<td>C</td>
<td>returns the unit of measure for the margin setting.</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 simply 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 do not usually need to concern
themselves with this value 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 _getData method call so that it does not continue to return rows or columns while measuring occurs.

### _getNumberOfColumns

**Returns the number of columns in the data vector**

#### Syntax

CALL SEND (data-vector-id, _getNumberOfColumns, num-columns);

#### Argument Type Description

<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 in the data vector</td>
</tr>
</tbody>
</table>

#### Details

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

### _getPrinting

**Returns the printing status**

#### Syntax

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

#### Argument Type Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printing</td>
<td>C</td>
<td>returns 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 _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 _getData 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);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>protect</td>
<td>C</td>
<td>returns 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 _getProtect method is valid if the viewer is a form editor or a table editor.

### _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.
**_getRow**

Returns the row coordinates

---

**Syntax**

CALL SEND (data-vector-id, '_getRow', row);

<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 identifier of an SCL list to contain the row coordinates.</td>
</tr>
</tbody>
</table>

**Details**

The `_getRow` method is valid if the viewer is a form editor or a table editor.

---

**_getText**

Returns the text

---

**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 formatted text to be displayed in the viewer</td>
</tr>
</tbody>
</table>

**Details**

The `_getText` method is valid only if the viewer is a table editor. If the viewer is a form editor, use `_getUnformattedText` or `_getUnformattedValue`.

---

**_getType**

Returns the type

---

**Syntax**

CALL SEND (data-vector-id, '_getType', type);
_getUnformattedValue

Returns the unformatted numeric value

Syntax
CALL SEND (data-vector-id, '_getUnformattedValue', value);

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

Details
The _getUnformattedValue method is valid only if the viewer is a form editor. If the viewer is a table editor, use the _getText method.
getUserError

Returns the user error state

Syntax

CALL SEND (data-vector-id, '_getUserError', error);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error</td>
<td>C</td>
<td>returns the error state</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y'   a user error has occurred in this column</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N'   a user error has not occurred in this column</td>
</tr>
</tbody>
</table>

Details

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

The default error state is 'N'. The user error state is set to 'Y' if _erroronColumn is called on the model for this column. This way of indicating errors differs from a model validation or data form data widget error. For more information on those types of errors, see the _getInternalError and _setInternalError methods in this class.

getVjust

Returns the vertical justification

Syntax

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>just</td>
<td>C</td>
<td>returns 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 _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>);

<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 wraps the text over multiple lines. For example, if split-char is set to the backslash (\) character, the text `One\Two\Three` will be split over 3 lines.

---

**init**

Initializes 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 to contain initialization information that the data vector can use when initializing</td>
</tr>
</tbody>
</table>

**Details**

The _init method allocates the GET_DATA data vector resources. It also calls the super _init method. The _init method is called whenever you make an instance of the GET_DATA Data Vector class, either through a call to _new or to the instance function.
The data-vector-info list is optional. The list tells the data vector which type of viewer this data vector will be used with if that information is known. This list can also contain NOFORME or NOTABLEE, which are numeric named items.

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 GET_DATA data vector, if you know that it will only be used in conjunction with the table editor, you can save resources 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 will be allocated, and when you attempt to call methods of the GET_DATA Data Vector class that deal with form-specific information, you will receive a program halt.

On the other hand, when you create the GET_DATA 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 will not be any resource savings, but when you attempt to call methods of the Row/Column Data Vector class that deal with table-specific information, you will 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 is a code fragment that creates an instance of the GET_DATA Data Vector class for use only with an instance of the Form Editor class. If this code fragment were used as part of an application, any objects that were instantiated during execution would need to be terminated when those objects were no longer needed. Likewise, any lists that were created during execution would need to be deleted when those lists were no longer needed.

Create an instance of the GET_DATA Data Vector class for form only.

```
arglist = makelist();
rc = setnitemn(arglist, 1,
   'NOTABLEE');
classid = loadclass('sashelp.fsp.gddvec');
call send(classid, '_new', gddvecid,
   arglist);
call send(gddvecid, '_setNumberOfColumns',
   1);
call send(gddvecid, '_setIndex', 1);
```

Try to call a table-only method.

```
call send(gddvecid, '_setBorderColor',
   'TOP', 'green');
```
The above code causes a program halt because it attempts to call the \_setBackgroundColor method, which is only for the table, even though you specified that this instance will only be used with a form. The following error message is generated:

```
ERROR: Table-specific method called on form-specific data vector.
Arguments passed to SEND:
  1 GET_DATA Data Vector ID = 33
  2 (Character Literal) = \_setBorderColor
Parameters passed to SEND ENTRY:
  1 (Character Literal) = \texttt{\text quoted \`green\'}
```

---

\_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 a form editor or a table editor.

---

\_setBackgroundPattern

Sets the background pattern

Syntax

```
CALL SEND (data-vector-id, \_setBackgroundPattern\', pattern);
```
### _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'</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>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. For the border to be visible, it must have a width. You can query and set the border width with _getBorderWidth and _setBorderWidth.

### _setBorderStyle

**Sets the border style**

---

<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:</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 _setBackgroundPattern method is valid only if the viewer is a table editor.
**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>
<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>
</tbody>
</table>

| style    | C    | specifies the border style: |
|          |      | 'SOLID' solid line |
|          |      | 'DOTTED' dotted line |
|          |      | 'DASHED' dashed line |

**Details**

The _setBorderStyle method is valid only if the viewer is a table editor. For the border to be visible, it must have a width. You can query and set the border width with _getBorderWidth and _setBorderWidth.

---

**_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>
</tbody>
</table>
The _setBorderWidth method is valid only if the viewer is a table editor.

### _setClassName

Sets the data class name

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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 with unit of measure.</td>
</tr>
</tbody>
</table>

**Details**

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

The data class is the name of the widget class the form editor uses to display data. The instance variables list of this class is controlled with _getClassProperties and _setClassProperties.

Note that the form editor only uses the data class and data attributes specified by _setClassName and _setClassProperties for row number 0. The form editor does this because the form editor cannot have a different data class for each row. However, the form editor can have a different data class for each column. Therefore, you can only set information about the data class through these methods for the first _getData call that the form editor makes.

### _setClassProperties

Sets the data attributes list

<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 data class name</td>
</tr>
</tbody>
</table>

**Details**

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

The data class is the name of the widget class the form editor uses to display data. The instance variables list of this class is controlled with _getClassProperties and _setClassProperties.

Note that the form editor only uses the data class and data attributes specified by _setClassName and _setClassProperties for row number 0. The form editor does this because the form editor cannot have a different data class for each row. However, the form editor can have a different data class for each column. Therefore, you can only set information about the data class through these methods for the first _getData call that the form editor makes.

### Syntax

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

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

Sets the column coordinates for the GET_DATA data vector

#### Syntax

```plaintext
CALL SEND (data-vector-id, '_setColumn', column);
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>column</td>
<td>C</td>
<td>specifies the column coordinates.</td>
</tr>
</tbody>
</table>

### _setColor

Sets the foreground color

#### Syntax

```plaintext
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 _setClassProperties method is valid only if the viewer is a form editor.

The attributes list is the instance variable list of the data class used by the form editor. For example, if the form editor displays its data with the Text Entry class, the attributes list will reflect the instance variables list of the Text Entry class. You control the class used by the form editor with _getClassName and _setClassName.

Note that the form editor only uses the data class and data attributes specified by _setClassProperties for row number 0. The form editor does this because the form editor cannot have a different data class for each row. However, the form editor can have a different data class for each column. Therefore, you can only set information about the data class through these methods for the first _getData call that the form editor makes.

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

### _setColumn

Sets the column coordinates for the GET_DATA data vector

#### Syntax

```plaintext
CALL SEND (data-vector-id, '_setColumn', column);
```
**_setFont**

Sets the font

---

**Syntax**

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

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

**Details**

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

The list sent to _setFont should come from either 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 GET_DATA data vector to save the font and then call _setFont to set that font on another GET_DATA data vector.

---

**_setFormatLength**

Sets the format length

---

**Syntax**

CALL SEND (data-vector-id, '_setFormatLength', length);
Details
The `_setFormatLength` method is valid if the viewer is a form editor or a table editor. The value specified for `_setFormatLength` is the maximum number of characters that the viewer can display.

__`_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>returns 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>
<tr>
<td></td>
<td></td>
<td>'NONE' (The model does not specify justification; therefore, use the viewer's default specification.)</td>
</tr>
</tbody>
</table>

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

__`_setIndex`__

Sets the column index into the data vector

**Syntax**

CALL SEND (data-vector-id, '_setIndex', index);
### _setIndex

**Argument Type Description**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>N</td>
<td>specifies the column index</td>
</tr>
</tbody>
</table>

**Details**

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

You should call _setIndex to tell the GET_DATA data vector to which column index (out of the number of columns specified by _setNumberOfColumns) subsequent method calls refer.

The column index reflects the GET_DATA data vector’s idea of the current column. However, the column index may not reflect the display order, which is set with _setCurrentColumn.

See “Calling Methods of the GET_DATA Data Vector Class” on page 1352 for more information.

### _setInherit

**Sets the inherit status**

**Syntax**

CALL SEND (data-vector-id, '_setInherit', inherit);

**Argument Type Description**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inherit</td>
<td>C</td>
<td>specifies the inherit status:</td>
</tr>
</tbody>
</table>

- 'Y' the data class inherits its attributes from the data class rather than from the settings in the form editor attributes screens
- 'N' the data 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 the inherit status is set to 'Y', the colors, fonts, and other attributes specified under the form attributes screens are ignored in favor of the attributes set in the specified data class.

### _setInternalError

**Sets the internal error state**
Syntax
CALL SEND (data-vector-id, '_setInternalError', error);

Argument | Type | Description
--- | --- | ---
error | C | sets the error state:
'Y' | an internal error has occurred in this column
'N' | an internal error has not occurred in this column

Details
The _setInternalError method is valid only if the viewer is a form editor. The default error state is 'N'. The internal error state is set to 'Y' if an error occurs in the data widget (such as an invalid type) or if a validation error occurs in the model (such as a value that exceeds the maximum). This way of indicating errors differs from a call to _erroronColumn. For more information on those types of errors, see the _getUserError and _setUserError methods in this class.

_setLightSource
Sets the light source for button borders

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

Where ... | Type | Description
--- | --- | ---
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);

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 unit of measure for the margin setting.

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
---|---|---
measuring | C | specifies the measuring status:
  - 'Y' the viewer is measuring the data
  - 'N' the viewer is not measuring the data

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 _setPrinting and _getPrinting, 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
_getData method call so that it does not continue to return rows or columns while measuring is occurring.

_setNumberOfColumns

Sets the number of columns in the data vector

Syntax

CALL SEND (data-vector-id, '_setNumberOfColumns', num-columns<, return-code>);

<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 in the GET_DATA data vector</td>
</tr>
<tr>
<td>return-code</td>
<td>N</td>
<td>returns a code that designates whether the _setNumberOfColumns call is successful:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 successful</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;0 unsuccessful</td>
</tr>
</tbody>
</table>

Details

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

After the GET_DATA data vector is created, _setNumberOfColumns should be called immediately to tell the _getDATA data vector how many columns that instance refers to. This is necessary because _getData and _setData can deal with more than one column at a time.

_setPrinting

Sets the printing status

Syntax

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

CALL SEND (data-vector-id, '_setNumberOfColumns', num-columns<, return-code>);
### _setprinting

**Argument**
- **printing**: C

**Description**
- Specifies the printing status:
  - 'Y': the viewer is printing the data
  - 'N': the viewer is not printing the data

**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 _setMeasuring and _getMeasuring, 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 _getData 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);

**Argument**
- **protect**: C

**Description**
- Specifies the protect status:
  - 'Y': protected
  - 'N': not protected

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

### _setReverse

**Sets the reverse status**

**Syntax**

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

**Description**

 Specifies the formatted text to be displayed in the viewer.

**Details**

 The `_setText` method is valid only if the viewer is a table editor. If the viewer is a form editor, use `_setUnformattedText` or `_setUnformattedValue`.

#### Argument Type Description

<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 formatted text to be displayed in the viewer</td>
</tr>
</tbody>
</table>

### _setRow Method

**Description**

 Sets the row coordinates.

**Syntax**

```
CALL SEND (data-vector-id, '_setRow', row);
```

#### Argument Type Description

<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 identifier of an SCL list to contain the row coordinates</td>
</tr>
</tbody>
</table>

**Details**

 The `_setRow` method is valid if the viewer is a form editor or a table editor.

### _setReverse Method

**Description**

 Specifies the reverse status:

 - 'Y' swap foreground and background colors
 - 'N' do not swap foreground and background colors

**Details**

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

Sets the unformatted text

Syntax
CALL SEND (data-vector-id, '_setUnformattedText', 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 unformatted text value</td>
</tr>
</tbody>
</table>

Details
The _setUnformattedText method is valid only if the viewer is a form editor. If the viewer is a table editor, use _setText.

_setUnformattedValue

Sets the unformatted numeric value

Syntax
CALL SEND (data-vector-id, '_setUnformattedValue', value);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>N</td>
<td>specifies the unformatted numeric value</td>
</tr>
</tbody>
</table>

Details
The _setUnformattedValue method can only be used if the viewer is a form editor. If the viewer is a table editor, use _setText.

_setUserError

Sets the user error state

Syntax
CALL SEND (data-vector-id, '_setUserError', error);
### _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 text-wrapping</td>
</tr>
<tr>
<td>split-char</td>
<td>C</td>
<td>split character for wrapping</td>
</tr>
<tr>
<td>Argument</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>wrapping</td>
<td>C</td>
<td>specifies the wrapping status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N'</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` wraps the text over multiple lines. For example, if `split-char` is set to the backslash (\) character, the text `One\Two\Three` will be split over three lines.