



CHAPTER

68

Form Editor Class

<i>Overview</i>	1283
<i>Using the Form Editor Class</i>	1284
<i>Making a Form Editor</i>	1284
<i>Using a Form Editor Object</i>	1284
<i>Methods</i>	1290
<i>Dictionary</i>	1290

Overview

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

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

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

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

The Form Editor class provides the ability to view and interact with a single row of data from a given data model using SAS/AF software widgets to represent the data model's columns. The Form Editor class uses the model as a data source. This model is expected to be a subclass of the Form Data Model class, which is the abstract class that defines the communication protocol between the Form Editor class and the model. For example, data form is a subclass of the form editor, and it uses Data Set Data Model, which is a subclass of the Form Data Model. This relationship is already set up for you, and no subclassing is involved. For more information, see the Form Data Model class.

Once attached to a model, the Form Editor class creates a default layout using a label and data widget to represent each column of the model. If the model contains more columns than will fit into the form editor's region, then the contents of the form editor are partitioned into units called pages. By default, each page contains as many widgets as will fit into the form editor's region.

After the creation of the layout, you can execute the frame, and then the column data from the data model are displayed in the associated widgets. During run mode, the form editor provides the basic row and page navigation actions via its pop-up menu. This separation of data and view is known as the Model-View-Controller (MVC) paradigm.

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

Parent:

sashelp.fsp.widget.class

Class:

sashelp.fsp.Form_e.class

Using the Form Editor Class

Making a Form Editor

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

- From a command line, issue the command

```
rm make 'form editor'
```

- Add the Form Editor class to the selection list of choices displayed by the **Make** action in the frame pop-up menu. Then you can create the object by selecting **Form Editor** from the list.
 - If you are using your own RESOURCE entry, edit it and add the Form Editor class. Make sure the DISPLAY status is on. **Form Editor** will be added to the selection list.
 - Otherwise, copy the default RESOURCE entry, SASHELP.FSP.BUILD.RESOURCE, into your current catalog. Edit the RESOURCE entry and make sure the DISPLAY status is on. **Form Editor** will be added to the selection list.

Using a Form Editor Object

Layouts

In a Form Editor, a layout consists of widgets which allow a user to interact with the data from the attached model. As the developer, you can control how the user enters and views the data by displaying the data using any widget available in a FRAME entry. Layouts can be as simple as a default layout or more complex.

The Form Editor allows the flexibility to create any type of layout that your particular application needs using all of the normal FRAME operations, such as, make, move, and resize.

Warning: SAS guidelines prohibit nesting this deep Creating a New(Default) Layout

You can create a new layout in one of several ways:

- Create a new Form Editor or subclass of a Form Editor object and specify a Data Class into the Attribute window. Once the Data Class has been assigned, associate a data source to that Data class in the Data Class Attributes window (under **Additional Attributes**). For example, if the Data Class was Data Set Data Model, then an appropriate data source would be the table SASUSER.CLASS. At

this point you can close the attribute window and a default layout will be created for you using all of the columns from the attached model. By default, input field widgets are used to represent the columns.

- Create a new Data Form object and specify a table (data set), such as SASUSER.CLASS. From the Data Form Attribute window, you can select **Customize Columns** to have more control over which columns will be placed automatically in your new layout and which widgets will be used to represent the columns. After completing the customizations in the attribute screen, end out of the attribute screen and a default layout will be created for you using the customizations you specified.

If you have customized your display but want to return to the default layout, you can start all over again by selecting the **Refill Using Attributes** pop-up menu item. This creates a whole new Form Editor or Data Form object but uses the current attribute settings from the Attribute Screen.

No matter how you create a new layout, you can add, delete, or modify any of the widgets in the layout at any time to customize the layout to your particular application. When using a Data Form, you can start from a blank layout by hiding all of the columns in the Customize Columns attribute window and then create and arrange the widgets inside the layout as needed. To save your customizations, you must specify a DATAFORM entry.

When creating a new layout, a default set of attributes is used to create the layout. These attributes are controlled using either the Form Editor Options window or by calling methods on the Form Editor. The following is a list of methods that control the appearance of a new (default) layout and are called automatically for you:

```

_setAttributes
_setDataBackgroundColor
_setDataClass
_setDataColor
_setDataFont
_setLabelBackgroundColor
_setLabelClass
_setLabelColor
_setLabelFont
_setLabelRepresentation

```

For information about any of these methods, see the specific method description.

Once a default layout has been created, you can execute the frame and the data widgets inside of the Form Editor will display the associated model column values for the current row. The default layout has many uses. For example, you can use the default layout as a starting point for the design of your particular layout and customize it to fit your needs. Another example is to use a Data Form to display different tables in run-mode. Each time a new table is displayed in the Data Form a new default layout could be created to represent the column from the new table.

Warning: SAS guidelines prohibit nesting this deep Widget Representations

When creating a default layout or when dragging a column from the Column Window onto a Form Editor, a model column is represented by a combination of widgets, known as a *widget representation*. There are four ways a widget representation can be made

and each is described below. In the descriptions, the widget that displays the column name or label is referred to as the *label widget*, and the widget that displays and accepts a value for the column is referred to as the *data widget*.

Two widgets (the default)

By default, the `Pair label/data widgets` option in the Form Editor Options window is deselected and the `GROUP_WIDGETS` attribute of the Form Editor is set to 'N'. This representation contains a label widget and a data widget.

Three widgets

To use this representation, select the `Pair label/data widgets` option in the Form Editor Options window or call the `_setAttributes` method setting the `GROUP_WIDGETS` attribute to 'Y'. Then call the `_setLabelRepresentation` method passing the argument of 'WIDGET'. This representation contains

- a label widget
- a data widget
- a container box that surrounds both the label and data widgets. This container box enables you to move both widgets together keeping the distance between the two widgets consistent.

One widget with a Region Title

To use this representation, deselect the `Pair label/data widgets` option in the Form Editor Options window or call the `_setAttributes` method setting the `GROUP_WIDGETS` attribute to 'N'. Then call the `_setLabelRepresentation` method passing an argument of 'REGTITLE'. This representation contains a data widget that displays the linked column's name or label as the region title.

One widget

To use this representation, deselect the `Pair label/data widgets` option in the Form Editor Options window or call the `_setAttributes` method setting the `GROUP_WIDGETS` attribute to 'N'. Then call the `_setLabelRepresentation` method passing an argument of 'NOLABEL'. This representation contains only a data widget and does not contain the column name or label.

Warning: SAS guidelines prohibit nesting this deep Placement of Widget Representations in a Default Layout

When creating a default layout, placement of the widget representations can be either vertical (the default) or horizontal. This order can be set in the Form Editor attribute window or by calling the `_setAttributes` method.

Warning: SAS guidelines prohibit nesting this deep Modifying a Layout

After creating a new layout using one of the techniques mentioned in "Creating a New(Default) Layout" on page 1284, you can modify that layout using all of the normal FRAME operations. The only limitation is that widgets cannot be moved or copied into or out of a Form Editor.

In order to make any customizations to a layout, you must specify a DATAFORM entry in the Form Editor's Attribute window. Without a DATAFORM entry, your customizations are not available when you execute the frame and are lost when you end from the frame.

The Form Editor allows you to create a custom layout quickly and easily without requiring an SCL program. Since the Form Editor allows you to represent a model column as a widget, you can take advantage of special features of the widgets. They can

perform self-validation of values, allow the user to enter values faster and more reliably, and also enhance the appearance of a layout.

These changes made to the default layout to create the custom layout below were easy to make and improved the look and feel of the frame. Also, this customized layout has multiple pages to help split up the information so that it is not so overwhelming to the user.

Data and Label Widgets

The widgets inside a Form Editor can be divided into three categories:

Unlinked Widgets

Widgets that are not linked to a model column.

Data Widgets

Widgets that are linked to model columns that allow a user to view and modify the column's value. Data widgets can be created at any time during the construction of a layout by creating new widget representations or transforming an unlinked widget into a data widget by linking a model column to it. By default the class used to create this widget is an input field. You can customize this by calling the `_setDataClass` methods.

Label Widgets

Widgets that display either the name or label of a model column. These widgets are initially linked to a model column and can only be created when a new widget representation is made. When a widget representation is made, an internal association is made between the data and label widgets. When the column name or label of the associated data widget changes, that change is reflected in the label widget. By default, the class used to create the label widget is an input field label. If you are using a data form object, you can customize this by calling the `_setLabelClass` method or through the data form object's attribute window.

When data and label widgets are internally linked, we will use the text of the column name or column label for the text of the label widget instead of any changes to the text you make at the widget level. This also applies if you display the label text as a region title.

Warning: SAS guidelines prohibit nesting this deep Attributes for the Label and Data Widgets

The Form Editor calls the following methods to get and set attributes of both the label and data widgets.

```

_getBackgroundColor
_setBackgroundColor
_getColor
_setColor
_getFont
_setFont
_getFormat
_setFormat
_getInformat
_setInformat

```

`_getJustify`

`_setJustify`

If a method is not defined for a particular widget class, then that attribute is neither set nor retrieved. The attributes that are set for the data and label widgets are controlled by the model attached to the Form Editor. A model can have a set of attribute values that are specific to a particular column. If the column linked to a data or label widget has any attribute values specifically set, then those values take precedence over the default attribute values of the Form Editor. For example, if a model column has a background color specified, then that color is used instead of the default background color of the Form Editor. Otherwise, the Form Editor default attribute value will be used.

The Form Editor has default attribute values for the Data background and foreground color and font as well as similar settings for the Label. The Form Editor does not have default settings for the format, informat, or justification attributes, so these attributes are not set unless the model specifies them. The `_get` methods listed are used to query the attributes from the data or label widgets that you have set in the widget's attribute window.

Note: The Form Editor calls the `_setData` method to set the widget's value to the value of the linked model column. The Form Editor calls the `_getData` method to retrieve the widget value modified by the user to be stored in the linked model column. These methods are called on all data widgets and are crucial for every data widget. Without the `_setData` and `_getData` methods, the data widgets could not display or retrieve changes made to linked model columns. The widgets supplied by SAS/AF software each have a `_setData` and `_getData` method defined, but if you create your own widget subclass, you must define a `_getData` and a `_setData` method for it in order to use that widget subclass in a data form. Δ

Pages

The contents of a Form Editor layout can be split into units called *pages*. If a model has more columns than will fit on the initial page, other pages are added to display those model columns. You may also decide to add additional pages. Pages allow you to

- split your layout into smaller groups to make your application easier to use. The groups could be based on column importance or there could be a logical grouping of columns that naturally would go together on a page.
- reduce the need to scroll within one page. If you use multiple pages rather than one long page, you can use navigation features (pop-up menus, earmarks, etc.) to jump to different pages to get to the columns you need. If you use one long page, you must scroll through the page to get to the columns you need.

The Form Editor has no limit on the number of pages it can have, nor is there a limit on the number of widgets per page. You are only limited by the resources available on your system. A model column may be linked to a widget on more than one page. But realistically, you should limit the number of pages and widgets per pages since the more pages and widgets per page you have, the longer it takes for the frame to display.

The scroll bars, when active, allow you to scroll within the current page, not between pages or rows. Because the scroll bars scroll within a page, the page can be as large as you like. During the creation of a default layout, the page size is set to the size of the Form Editor's region size. This size can be changed by scrolling down or to the right and adding widgets. Any page customizations that are done, are stored in the DATAFORM entry. DATAFORM entries can be used interchangeably between Form Editors; so if a new Form Editor is smaller than the original Form Editor, the scroll bars will allow you to view the entire contents of the pages.

During build mode, the scroll bars have no boundary, which enables you to place widgets anywhere within a page. During run mode, scroll bars enable you to scroll only to the lowest and right-most widget on the current page. For example, the vertical scroll bar is maximized because all of the widgets are visible in the Form Editor's region. The horizontal scroll bar allows you to scroll to the right and view the entire contents of the page.

Warning: SAS guidelines prohibit nesting this deep Adding and Deleting Pages

All Form Editor layouts have at least one page. The Form Editor allows you to add pages via the build mode pop-up menu items, **Add page after current** and **Add page before current**. After selecting one of these pop-up menu items, you are positioned on the new empty page. After a page is added, all of the pages are renumbered and now are referenced using the new number.

You can delete any page from a layout by selecting the build mode pop-up menu item **Delete Page...** You will be prompted with a dialog box to verify that you want to delete this page. When a page is deleted, all widgets that were contained on that page are deleted also. The page that has been deleted cannot be recovered unless it was previously stored in a DATAFORM entry and you cancel out of the frame. By cancelling out of the frame, you also lose other changes that you have made, so be careful when deleting pages. After a page is deleted, all of the remaining pages are renumbered and now are referenced by using the new numbers. You cannot delete the last page remaining in a layout.

Warning: SAS guidelines prohibit nesting this deep Navigating Between Pages

When a Form Editor layout has multiple pages, users can navigate between the pages in order to view and enter the data. There are several ways that this can be done:

Run mode pop-up menu

There are four items on this menu that allow you to navigate between the pages of a layout, **First Page**, **Previous Page**, **Next Page**, and **Last Page**.

Earmarks

Earmarks make the layout look as though it has a turned down page in either the upper left, the upper right, or both corners. An earmark shown in the upper right corner indicates that there are additional pages. An earmark shown in the upper left corner indicates that there are previous pages. By double clicking on the left earmark, you will be taken to the first page. By double clicking on the right earmark, you will be taken to the last page.

You can control the display of earmarks from the Form Editor Options attributes window or by calling the `_setAttributes` method and passing an item of EARMARK with a value of 'Y' or 'N'. The color of the earmarks can also be controlled from the attribute screen or by calling `_setEarmarkColor` or `_setEarmarkOutlineColor`.

Warning: SAS guidelines prohibit nesting this deep Moving Widget Between Pages in a Form Editor

The Form Editor gives you the ability to move widgets between pages using the build-mode pop-up menu. To move a widget across pages in a form editor, follow these steps:

- 1 Go to the page of the data form where the widget is displayed.
- 2 Select the widget.
- 3 Select **Cut to form buffer** from the pop-up menu for the widget.

- 4 Go to the page where you want to paste the widget.
- 5 Select **Paste from form buffer** to paste the widget on the new page.

You cannot select multiple, individual widgets to move, but you can move a group of widgets by wrapping them in a container box and moving the container box.

Effect of Protecting Columns

You can prevent users from modifying a column or columns by "protecting" the column or columns. Columns can be protected by using the `_protectColumn` method or by setting the `Protected` attribute for that column or columns. The effect of protecting a column in a data form, form editor, or subclass of the form editor, is that you can type into columns that are protected, but the widgets revert to their previous state when you press enter. In addition, you cannot tab to or tab out of a protected column and labeled sections for protected columns do not run.

When a form editor is in browse mode, all of the columns are protected.

Methods

Methods specific to the Form Editor class are described here. Inherited methods are described in the Widget class. Any methods not understood by the Form Editor class are delegated to the attached data model.

Dictionary

`_attach`

Associates the specified data model with the form editor

Syntax

```
CALL SEND (form-id, '_attach', model-id);
```

Details

The `_attach` method associates the specified data object to the form editor. The data object is referred to as *the model* and is used as the data source for the form editor. The data model that is attached must be a subclass of the Form Data Model class (for example the Data Set Data Model class), which is the abstract class that defines the communication protocol between the Form Editor class and its attached model.

Because only one model can be attached to the viewer at a time, calling `_attach` on the viewer will detach any previously attached model.

See Also

`_detach`.

`_columnWindowInit`

Initializes the Column window

Syntax

```
CALL SEND (form-id, '_columnWindowInit');
```

Details

The `_columnWindowInit` method displays the Column Window if it is not already displayed or moves the Column Window to the foreground if it is already displayed.

This method is valid only during build mode and has no effect on the object during run mode.

`_columnWindowTerm`

Terminates the Column window

Syntax

```
CALL SEND (form-id, '_columnWindowTerm');
```

Details

The `_columnWindowTerm` method terminates the Column Window if only one form editor widget is present. Otherwise it notifies a form editor widget that the Column Window is no longer displayed.

This method is valid only during build mode and has no effect on the object in run mode.

`_detach`

Disassociates the currently attached data model

Syntax

```
CALL SEND (form-id, '_detach');
```

Details

The `_detach` method disassociates the currently attached data model from the form editor. When the form editor is detached from the model, all of the widgets that are contained in the form editor are deleted. `_detach` has no effect on the model itself; that is, the model is not deleted (terminated with `_term`).

`_getAttributes`

Returns the current attribute settings for the form

Syntax

```
CALL SEND (form-id, '_getAttributes', attr-list);
```

Argument	Type	Description
<i>attr-list</i>	N	specifies the identifier of an SCL list to contain the current values of all the named attributes listed in the table provided with the <code>_setAttributes</code> method.

See Also

`_setAttributes`.

`_getCurrentPage`

Returns the number of the displayed page

Syntax

```
CALL SEND (form-id, '_getCurrentPage', page-num);
```

Argument	Type	Description
<i>page-num</i>	N	returns the number of the displayed page

Details

The `_getCurrentPage` method always returns a number greater than 0 because the form editor always has at least one page defined.

`_getCurrentWidget`

Returns the object identifier of the active widget

Syntax

CALL SEND (*form-id*, '`_getCurrentWidget`', *widget-id*);

Argument	Type	Description
<i>widget-id</i>	N	returns the object identifier of the active widget

Details

If the active widget is not inside of the form editor or there is no active widget, then *widget-id* is set to 0.

`_getCurrentWidgetInfo`

Returns information about the active widget

Syntax

CALL SEND (*form-id*, '`_getCurrentWidgetInfo`', *info-lst*);

Argument	Type	Description
<i>info-lst</i>	N	specifies the identifier of an SCL list to contain information about the active widget

Details

The `_getCurrentWidgetInfo` method returns the widget identifier, the linked column name, and the linked column identifier of the active widget, as shown in the following table.

Table 68.1 Widget Information List Items

Item	Type	Value
'WIDGET_ID'	N	0
'COLUMN_NAME'	C	' '
'COLUMN_ID'	N	0

`_getDataBackgroundColor`

Returns default background color for the data widget

Syntax

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

Argument	Type	Description
<i>color-name</i>	C	returns the color name

Details

The `_getDataBackgroundColor` method always returns the default color that the form assigns to columns, regardless of whether you have set an alternate color for a column.

`_getDataClass`

Returns the default widget class and attributes for the data widget

Syntax

CALL SEND (*form-id*, '`_getDataClass`', *class-name*<, *class-attr*>);

Argument	Type	Description
<i>class-name</i>	C	returns the name of the data widget class
<i>class-attr</i>	N	specifies the identifier of an SCL list to contain the attributes used to create a widget of type <i>class-name</i>

Details

The `_getDataClass` method always returns the default widget class and attributes for the data widget, regardless of whether you have class or attributes settings for a

particular column. The `_getDataClass` method returns the default data widget class and attributes that are used when you create the initial display, after you call the `_refillUsingAttributes` method, or when you create a new widget representation using the Column Window.

`_getDataColor`

Returns the default foreground color for the data widget

Syntax

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

Argument	Type	Description
<i>color-name</i>	C	returns the color name

Details

The `_getData_color` method always returns the default color that the form assigns to columns, regardless of whether you have set an alternate color for a column.

`_getDataFont`

Returns the default font for the data widget

Syntax

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

Argument	Type	Description
<i>font-list-id</i>	N	specifies the identifier of an SCL list to contain the font attributes

Details

The `_getData_font` method always returns the default font that the form assigns to data widgets, regardless of whether you have set an alternate font for a data widget. You should not directly manipulate the contents of the font list. Instead, send the list as a whole to any method that sets fonts, for example, `_setDataFont` or `_setLabelFont`. You can use the SCL FONTSEL function to create a new font list, if desired.

`_getEarmarkColor`

Returns the fill color used to draw the earmarks

Syntax

CALL SEND (*form-id*, '`_getEarmarkColor`', *color-name*);

Argument	Type	Description
<i>color-name</i>	C	returns the name of the color used to draw the earmarks. Earmarks are used to navigate between pages of the data form, form editor, or subclass of the form editor.

`_getEarmarkOutlineColor`

Returns the outline color used to draw the earmarks

Syntax

CALL SEND (*form-id*, '`_getEarmarkOutlineColor`', *color-name*);

Argument	Type	Description
<i>color-name</i>	C	returns the name of the color used to draw the outline of the earmarks. Earmarks are used to navigate between pages of the data form, form editor, or subclass of the form editor.

`_getHscroll`

Returns the default horizontal scroll unit

Syntax

CALL SEND (*form-id*, '`_getHscroll`', *units*<, *num-units*>);

Argument	Type	Description
<i>units</i>	C	returns the default horizontal scrolling unit
<i>num-units</i>	N	returns the default horizontal number of units to scroll

Details

You set the default scrolling values for *units* and *num-units* using the `_setHscroll` method. See `_setHscroll` for a list of units. Use horizontal scrolling to navigate through the pages of the form editor.

See Also

`_hscroll` and `_setHscroll`.

`_getLabelBackgroundColor`

Returns the default background color of the label widget

Syntax

CALL SEND (*form-id*, '`_getLabelBackgroundColor`', *color-name*);

Argument	Type	Description
<i>color-name</i>	C	returns the color name

Details

The `_getLabelBackgroundColor` method always returns the default color that the form assigns as the background color for label widgets, regardless of whether you have set an alternate background color for label widgets.

`_getLabelClass`

Returns the default widget class and attributes for the label widget

Syntax

CALL SEND (*form-id*, '`_getLabelClass`', *class-name*<, *class-attr*>);

Argument	Type	Description
<i>class-name</i>	C	returns the name of the label widget class
<i>class-attr</i>	N	specifies the identifier of an SCL list to contain the attributes used to create a widget of type <i>class-name</i>

Details

The `_getLabelClass` method always returns the default widget class and attributes for the label widget that the form assigns to label widgets, regardless of whether you have class or attribute settings for a particular column. The `_getLabelClass` method returns the default label widget class and attributes that are used when you create the initial display, after you call `_refillUsingAttributes`, or when you create a new widget representation using the Column Window.

`_getLabelColor`

Returns the default foreground color of the label widget

Syntax

CALL SEND (*form-id*, '`_getLabelColor`', *color-name*);

Argument	Type	Description
<i>color-name</i>	C	returns the color name

Details

The `_getLabelColor` method always returns the default color that the form assigns as the foreground color for label widgets, regardless of whether you have set an alternate foreground color for label widgets.

`_getLabelFont`

Returns the default font for the label widget

Syntax

CALL SEND (*form-id*, '`_getLabelFont`', *font-list-id*);

Argument	Type	Description
<i>font-list-id</i>	N	specifies the identifier of an SCL list to contain the font attributes

Details

You should not directly manipulate the contents of the font list. Instead, send the list as a whole to any method that sets fonts, for example, `_setDataFont` and `_setLabelFont`. You can use the SCL FONTSEL function to create a new font list, if desired. The `_setLabelFont` method always returns the default font that the form assigns as the font for label widgets, regardless of whether you have set an alternate font for label widgets.

`_getLabelRepresentation`

Returns the default representation of the label

Syntax

CALL SEND (*form-id*, '`_getLabelRepresentation`', *rep*);

Argument	Type	Description
<i>rep</i>	C	returns the representation of the label

Details

The `_getLabelRepresentation` method always returns the default representation of the label that the form assigns for label widgets, regardless of whether you have set an alternate representation for label widgets.

See Also

`_setLabelRepresentation`.

`_getLinkedColumn`

Returns the name of the column linked to the specified widget

Syntax

CALL SEND (*form-id*, '`_getLinkedColumn`', *widget-name*, *column-name*);

Argument	Type	Description
<i>widget-name</i>	C	specifies the name of the widget
<i>column-name</i>	C	returns the name of the column

Details

The `_getLinkedColumn` method returns the column that is linked to the widget with the name *widget-name*. If the requested widget does not have an associated column or the requested widget is not in the form editor, then *column-name* has a blank value.

`_getPageCount`

Returns the total number of pages

Syntax

CALL SEND (*form-id*, '`_getPageCount`', *page-count*);

Argument	Type	Description
<i>page-count</i>	N	returns the total number of pages

Details

The `_getPageCount` method always returns a number greater than 0, because there is always at least one page defined.

`_getProperties`

Returns a list of information that describes the state of the form and the attached model

Syntax

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

Where ...	Type	Description
<i>prop-list</i>	N	specifies the identifier of an SCL list to contain the properties of the form

Details

The `_getProperties` method is useful for copying or re-creating a form. You can pass this properties list to the `_new` method of a new form or the `_setProperties` method of an existing form. The properties list also includes information specific to the data model.

`_getVscroll`

Returns the default vertical scroll unit

Syntax

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

Argument	Type	Description
<i>units</i>	C	returns the default vertical scrolling unit
<i>num-units</i>	N	returns the default vertical number of units to scroll

Details

The default scrolling values for *units* and *num-units* are set using the `_setVscroll` method. See `_setVscroll` for a list of units.

See Also

`_setVscroll` and `_vscroll`.

`_getWidget`

Returns the widget identifier for the specified widget name

Syntax

CALL SEND (*form-id*, '`_getWidget`', *widget-name*, *widget-id*);

Argument	Type	Description
<i>widget-name</i>	C	specifies the name of the widget
<i>widget-id</i>	N	returns the object identifier for the specified widget

Details

If a value for *widget-name* is not found, then the *widget-id* is set to 0.

`_getWidgets`

Retrieves information about all of the form editor's widgets

Syntax

CALL SEND (*form-id*, '`_getWidgets`', *widget-list*);

Argument	Type	Description
<i>widget-list</i>	N	specifies the identifier of an SCL list to contain the information about the form editor's widgets

Details

The `_getWidgets` method returns complete information about all of the widgets inside of the form editor and places them in the list as sublist items. Each sublist in the list is named using the object's name.

`_gotoColumn`

Displays and activates the widget associated with the requested column

Syntax

CALL SEND (*form-id*, '`_gotoColumn`', *column-name*<, *occurrence*>);

Argument	Type	Description
<i>column-name</i>	C	specifies the name of the column
<i>occurrence</i>	N	specifies the occurrence to display

Details

The `_gotoColumn` method ensures that the *n*th widget, starting from the first page and working down to the last page, associated with *column-name* is displayed and becomes the active widget. If the *n*th *occurrence* of an associated widget does not exist, the method causes a program halt.

`_gotoPage`

Displays the requested page

Syntax

CALL SEND (*form-id*, '`_gotoPage`', *page-num*);

Argument	Type	Description
<i>page-num</i>	N	specifies the page number to display. Valid values range from 1 to the maximum page count. Reserved numbers that perform routine actions are <ul style="list-style-type: none"> -1 go to the first page -2 go to the previous page -3 go to the next page -4 go to the last page

Details

A program halt occurs if the requested page is out of range (except for the reserved numbers listed in the previous table).

`_gotoRowNumber`

Displays the requested row

Syntax

CALL SEND (*form-id*, '`_gotoRowNumber`', *row-num*);

Argument	Type	Description
<i>row-num</i>	N	specifies the row to display. Reserved numbers that perform routine actions are -1 go to the first row -2 go to the previous row -3 go to the next row -4 go to the last row

Details

The `_gotoRowNumber` method requests the form editor to display a different row. If the current row is in error, then the form editor does not honor the request to move to the requested row. If the form editor and its children are not in error, then the form editor moves to the requested row and populates all of the linked widgets with their values.

`_hscroll`

Scrolls the pages of the form editor

Syntax

CALL SEND (*form-id*, '`_hscroll`'<, *units*<, *num-units*>>);

Argument	Type	Description
<i>units</i>	C	specifies the unit by which the form scrolls. The default scroll unit of PAGE is used if you do not specify a value for <i>units</i> . See <code>_setHscroll</code> for a list of the valid values.
<i>num-units</i>	N	specifies the number of pages to scroll the form. For scrolling forward, specify a positive number. For scrolling backward, specify a negative number. The default scroll amount of 1 is used if you do not specify a value for <i>num-units</i> .

Details

You set the default values using the `_setHscroll` method.

See Also

`_getHscroll` and `_setHscroll`

`_popup`

Displays the run-mode pop-up menu

Syntax

CALL SEND (*form-id*, '_popup' *fill-lst*, *selection*);

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

Details

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

_refillUsingAttributes

Re-creates the form editor using the current attributes

Syntax

CALL SEND (*form-id*, '_refillUsingAttributes'<, *prompt*>);

Argument	Type	Description
<i>prompt</i>	C	specifies whether to prompt the user to continue with the refill process: <ul style="list-style-type: none"> 'Y' prompt the user (default) 'N' do not prompt the user

Details

The `_refillUsingAttributes` method completely wipes out the current contents of the form editor and refills it using the current attribute settings. Because a new default layout is created, the total number of pages in the form may change.

CAUTION:

This method is a destructive operation. Use `_refillUsingAttributes` when you want to completely start over as if you were creating a new form editor from scratch and a default layout were created for you. The default action prompts the user with a dialog box to allow the cancellation of the method. △

`_refreshPage`

Refreshes the values of the linked widgets on the requested page

Syntax

CALL SEND (*form-id*, '`_refreshPage`', *page-num*);

Where ...	Type	Description
<i>page-num</i>	N	specifies the number of the page to refresh.

Details

The `_refreshPage` method allows a user to force a refresh on a specific page. This refresh updates all of the linked widget values with the current column values from the locked row.

`_setAttributes`

Sets the attributes of the form

Syntax

CALL SEND (*object-id*, '`_setAttributes`', *attr-list*);

Argument	Type	Description
<i>attr-list</i>	N	specifies the identifier of an SCL list that contains any number of the named attributes listed in the table below.

Table 68.2 Attribute List

Attribute Name	Type	Default	Description
DISPLAY_columnWINDOW	C	'N'	indicates whether the Column Window will be displayed each time the frame is built.
GROUP_WIDGETS	C	'N'	indicates whether a container will be created to parent both the label and data widgets.
EARMARKS	C	'Y'	indicates whether the earmarks to navigate between pages will be displayed.

Attribute Name	Type	Default	Description
DROPSITE_NOTIFICATION	C	'Y'	indicates whether the user will be notified of a valid drop site by changing the objects border color.
ONE_PAGE	C	'N'	indicates whether only one page will be used when creating a default layout. If 'N', then multiple pages will be used.
EDIT	C	'N'	indicates whether the data from the attached model can be edited
WIDGET_PLACEMENT	C	'VERTICAL'	indicates how columns are ordered in the form. Possible values are 'HORIZONTAL' and 'VERTICAL'.

`_setCurrentWidget`

Sets the active widget

Syntax

CALL SEND (*form-id*, '`_setCurrentWidget`', *widget-id*);

Argument	Type	Description
<i>widget-id</i>	N	specifies the object identifier of the widget

Details

The `_setCurrentWidget` method makes the requested widget active if and only if it is contained inside the form editor. Otherwise, the method causes a program halt.

`_setDataBackgroundColor`

Sets the default background color for the data widget

Syntax

CALL SEND (*form-id*, '`_setDataBackgroundColor`', *color-name*);

Argument	Type	Description
<i>color-name</i>	C	specifies the color name

Details

The color set is the default color. This color may not be used if a specific color has been chosen for this area.

`_setDataClass`

Sets the default widget class and attributes for the data widget

Syntax

CALL SEND (*form-id*, '`_setDataClass`', *class-name*<, *class-attr*>);

Argument	Type	Description
<i>class-name</i>	C	specifies the name of the widget class
<i>class-attr</i>	N	specifies the identifier of an SCL list that contains the attributes that are used to create a widget of type <i>class-name</i>

Details

The class set is the default class for the data widget. This class may not be used if a specific class has been chosen for this data widget. The `_setDataClass` method sets the default widget class and attribute list that are used to create a data widget. Just calling this method has no direct effect on a form editor's layout. The new default class is used during the creation of an initial display, after a call to the `_refillUsingAttributes` method, or when you create a new widget representation using the Column Window.

`_setDataColor`

Sets the default foreground color for the data widget

Syntax

CALL SEND (*form-id*, '`_setDataColor`', *color-name*);

Argument	Type	Description
<i>color-name</i>	C	specifies the color name

Details

The color set is the default foreground color for the data widget. This color may not be used if a specific color has been chosen for this data widget.

`_setDataFont`

Sets the default font for the data widget

Syntax

CALL SEND (*form-id*, '`_setDataFont`', *font-list-id*);

Argument	Type	Description
<i>font-list-id</i>	N	specifies the identifier of an SCL list that contains the font information

Details

The font set is the default font for the data widget. This font may not be used if a specific font has been chosen for this data widget. The list you send to the `_setDataFont` method should come from either a method call to retrieve the font from an object that saves its font in a list format (for example, `_getDataFont`) or from the SCL FONTSEL function.

`_setEarmarkColor`

Sets the fill color used to draw the earmarks

Syntax

CALL SEND (*form-id*, '`_setEarmarkColor`', *color-name*);

Argument	Type	Description
<i>color-name</i>	C	specifies the name of the color used to draw the earmarks. Earmarks are used to navigate between pages of the data form, form editor, or subclass of the form editor.

`_setEarmarkOutlineColor`

Sets the outline color used to draw the earmarks

Syntax

CALL SEND (*form-id*, '`_setEarmarkOutlineColor`', *color-name*);

Argument	Type	Description
<i>color-name</i>	C	specifies the name of the color used to draw the outline of the earmarks. Earmarks are used to navigate between pages of the data form, form editor, or subclass of the form editor.

`_setHscroll`

Sets the default page scrolling units and number of units

Syntax

CALL SEND (*form-id*, '`_setHscroll`', *units*<, *num-units*>);

Argument	Type	Description
<i>units</i>	C	specifies the default unit by which a form editor scrolls pages: 'PAGE' scrolls either forward or backward the requested number of pages (default)

Argument	Type	Description
		'MAX' scrolls to the last page if a value for <i>num-units</i> is omitted or if the <i>num-units</i> value is positive. If the <i>num-units</i> value is negative, then it scrolls to the first page.
<i>num-units</i>	N	specifies the scrolling unit. A positive value scrolls forward; a negative number scrolls backwards.

Details

If the *num-units* parameter is not passed, then the default *num-units* value is 1. These values are used when you call `_hscroll` without parameters.

`_setLabelBackgroundColor`

Sets the default background color for the label widget

Syntax

CALL SEND (*form-id*, '`_setLabelBackgroundColor`', *color-name*);

Argument	Type	Description
<i>color-name</i>	C	specifies the color name

Details

The color set is the default color. This color may not be used if a specific color has been chosen for this area.

`_setLabelClass`

Sets the default widget class and attributes for the label widget

Syntax

CALL SEND (*form-id*, '`_setLabelClass`', *class-name*<, *class-attr*>);

Argument	Type	Description
<i>class-name</i>	C	specifies the name of the widget class
<i>class-attr</i>	N	specifies the identifier of an SCL list that contains the attributes that are used to create a widget of type <i>class-name</i>

Details

The class set is the default widget class for the label widget. This class may not be used if a specific class has been chosen for this label widget. The `_setLabelClass` method sets the default widget class and attribute list that are used to create a label widget. Just calling this method has no direct effect on a form editor's layout. The new default class is used during the creation of an initial display, after a call to `_refillUsingAttributes`, or when you create a new widget representation using the Column Window.

`_setLabelColor`

Sets the default foreground color for the label widget

Syntax

CALL SEND (*form-id*, '`_setLabelColor`', *color-name*);

Argument	Type	Description
<i>color-name</i>	C	specifies the color name

Details

The color set is the default foreground color for the label widget. This color may not be used if a specific color has been chosen for this label widget.

`_setLabelFont`

Sets the default font for the label widget

Syntax

CALL SEND (*form-id*, '`_setLabelFont`', *font-list-id*);

Argument	Type	Description
<i>font-list-id</i>	N	specifies the identifier of an SCL list that contains the font information

Details

The font set is the default font for the label widget. This font may not be used if a specific font has been chosen for this label widget. The list you send to the `_setLabelFont` method should come from either a method call to retrieve the font from an object that saves its font in a list format (for example, `_getDataFont`) or from the SCL FONTSEL function.

`_setLabelRepresentation`

Sets the default representation of the label

Syntax

CALL SEND (*form-id*, '`_setLabelRepresentation`', *rep*);

Argument	Type	Description
<i>rep</i>	C	specifies the representation of the label: <ul style="list-style-type: none"> 'WIDGET' displays the label as a separate widget 'REGTITLE' displays the label as the data widget's region title 'NOLABEL' displays no label

Details

The label representation set is the default label representation for the label widget. This representation may not be used if a specific representation has been chosen for this label widget. The `_setLabelRepresentation` method sets the default label representation that is used to create a label widget. Just calling this method has no direct effect on a form editor's layout. The new default representation is used during the creation of an initial display, after a call to the `_refillUsingAttributes` method, or when you create a new widget representation using the Column Window.

`_setLinkedColumn`

Links a widget to a particular column name

Syntax

CALL SEND (*form-id*, '_setLinkedColumn', *widget-name*<, *column-name*>);

Argument	Type	Description
<i>widget-name</i>	C	specifies the widget name
<i>column-name</i>	C	specifies the column name

Details

The `_setLinkedColumn` method associates the column from the attached data model with the widget inside of the form editor. Once associated, the widget displays the current value of the linked column in run mode. If the widget does not exist in the form editor or the column does not exist in the model, a program halt occurs. If a value for *column-name* is not passed, then the widget is disassociated from a column.

`_setMsg`

Specifies a message to be displayed on the containing frame's message line

Syntax

CALL SEND (*form-id*, '_setMsg', *msg*);

Where ...	Type	Description
<i>msg</i>	C	specifies the message to be displayed

Details

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

`_setProperties`

Restores the previous state of the form and the attached model

Syntax

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

Where ...	Type	Description
<i>prop-list</i>	N	specifies the identifier of an SCL list that is filled in by <code>_getProperties</code>

Details

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

`_setVscroll`

Sets the default row scrolling units and number of units

Syntax

```
CALL SEND (form-id, '_setVSCROLL', units<, num-units>);
```

Argument	Type	Description
<i>units</i>	C	specifies the default unit by which the form editor scrolls rows: <ul style="list-style-type: none"> 'ROW' scrolls either forward or backward the number of rows specified in <i>num-units</i> (default) 'MAX' scrolls to the last row if the value for <i>num-units</i> is omitted or if the <i>num-units</i> value is positive. If the <i>num-units</i> value is negative, then it scrolls to the first row.
<i>num-units</i>	N	specifies the default scrolling unit. A positive value scrolls forward; a negative number scrolls backwards.

Details

If the *num-units* parameter is not passed, then the default *num-units* value is 1. These values are used when you call the `_vscroll` method without parameters.

`_updateColumn`

Queries the model for one or more columns

Syntax

```
CALL SEND (form-id, '_updateColumn', col-lst);
```

Argument	Type	Description
<i>col-lst</i>	N	specifies the identifier of an SCL list that contains the column numbers that have changed

Details

This method is only useful if you are writing your own model. The `_updateColumn` method is called by the model to notify the form editor that some attribute about the passed columns has changed. The form editor should query the model for the new column attributes and should reflect the changed attributes in the associated widgets. The only column attributes that are not immediately honored are the widget class and the widget attributes. See the `_setDataClass` method for this class for more information about when class and attributes are used.

`_updateColumnDim`

Queries the data model for the number of columns

Syntax

```
CALL SEND (form-id, '_updateColumnDim');
```

Details

This method is only useful if you are writing your own model. The `_updateColumnDim` method is called by the model to notify the form editor that the number of defined columns in the model has changed.

`_updateHideColumns`

Hides one or more columns

Syntax

```
CALL SEND (form-id, '_updateHideColumns', col-lst);
```

Argument	Type	Description
<i>col-list</i>	N	specifies the identifier of an SCL list that contains the column numbers that should be hidden

Details

This method is only useful if you are writing your own model. The `_updateHideColumns` method is called by the attached model to notify the form editor that one or more columns are hidden from the model's point of view. The form editor hides the widgets linked to the columns in *col-list*.

`_updateProtectColumns`

Protects one or more columns

Syntax

CALL SEND (*form-id*, '_updateProtectColumns', *col-list*);

Argument	Type	Description
<i>col-list</i>	N	specifies the identifier of an SCL list that contains the column numbers that should be protected

Details

This method is only useful if you are writing your own model. The `_updateProtectColumns` method is called by the attached model to notify the form editor that one or more columns are protected from the model's point of view. The form editor protects the widgets linked to the columns in *col-list*.

`_updateRefill`

Queries the model for all columns' data and label information

Syntax

CALL SEND (*form-id*, '_updateRefill');

Details

This method is only useful if you are writing your own model. The `_updateRefill` method is called by the attached model to notify the form editor that it should

completely refill using the current attribute settings, and it should remove the current form editor layout.

`_updateRowData`

Queries the data model for the column values in the current row

Syntax

```
CALL SEND (form-id, '_updateRowData');
```

Details

This method is only useful if you are writing your own model. The `_updateRowData` method is called by the attached model to notify the form editor to requery the model for new column values for the displayed row.

`_updateUnhideColumns`

Unhides one or more columns

Syntax

```
CALL SEND (form-id, '_updateUnhideColumns', col-lst);
```

Argument	Type	Description
<i>col-lst</i>	N	specifies the identifier of an SCL list that contains the column numbers that should be unhidden

Details

This method is only useful if you are writing your own model. The `_updateUnhideColumns` method is called by the attached model to notify the form editor that one or more columns are unhidden from the model's point of view. The form editor unhides the widgets linked to the columns in *col-lst*.

`_updateUnprotectColumns`

Unprotects one or more columns

Syntax

CALL SEND (*form-id*, '_updateUnprotectColumns', *col-lst*);

Argument	Type	Description
<i>col-lst</i>	N	specifies the identifier of an SCL list that contains the column numbers that should be unprotected

Details

This method is only useful if you are writing your own model. The `_updateUnprotectColumns` method is called by the attached model to notify the form editor that one or more columns are unprotected from the model's point of view. The form editor unprotects the widgets linked to the columns in *col-lst*.

`_vscroll`

Scrolls the rows of the form editor

Syntax

CALL SEND (*form-id*, '_vscroll'<, *units*<, *num-units*>>);

Argument	Type	Description
<i>units</i>	C	specifies the unit by which the form scrolls
<i>num-units</i>	N	specifies the scrolling unit. A positive value scrolls forward; a negative number scrolls backwards.

Details

You set the default scrolling values for *units* and *num-units* using the `_setVscroll` method, which also describes the valid values.

When one or more widgets inside of the form editor are in error, the `_vscroll` method fails, which leaves you on the current row.

See Also

`_getVscroll` and `_setVscroll`.

