

AN INTRODUCTORY OVERVIEW OF THE DATA FORM AND DATA TABLE OBJECTS IN SAS/AF® FRAME ENTRIES

Ann Carpenter Rockett, SAS Institute Inc., Cary, NC

Abstract

The Data Form and Data Table objects in SAS/AF® FRAME entries allow you to view and edit data in a graphical manner with more customizations than ever allowed by the FSEDIT or FSVIEW procedures in SAS/FSP® software. This paper introduces you to Data Form and Data Table and discusses features of each object. You will learn how these objects can be customized, how they use a model and a viewer, and how they can take advantage of their own separate SCL entry.

Introduction

The Data Form and Data Table classes in SAS/AF FRAME entries are a graphical way to browse and edit data. These classes provide more customizations than ever allowed by the FSEDIT or FSVIEW procedures in SAS/FSP software.

Data Form

allows browsing or editing of a table (SAS data set) a single row (observation) at a time within a SAS/AF FRAME entry, using SAS/AF widgets to represent each column (variable).

Data Table

allows browsing or editing of your table in a tabular format within a SAS/AF FRAME entry. Each row represents an observation, and each column represents a variable in the table.

Both Data Form and Data Table share many features.

- They allow you to edit or browse data and refer to variables as columns and observations as rows. When you edit data using Data Form or Data Table, you are editing the data in place. That means no other SCL code is required to update the data.
- You can control the appearance of your data by specifying the use of labels for column headings instead of column names.
- You can add computed columns to your display and assign an SCL entry that is separate from your FRAME SCL. In this separate SCL entry, you can perform customized data validation and error handling.
- You can associate a DATAFORM entry with your Data Form or Data Table that allows you to save customized column attributes across sessions.

Data Form Features

By default, the Data Form object displays all columns using the input field widget. However, you can use any FRAME widget to represent your column. This allows you to take advantage of special characteristics of the widget you select when another widget gives more control. For example, if a column has a short list of discrete values, you can use a radio box to display the values to the user. In this case, only valid values are shown so you do not need to perform error checking on this column.

Inventory Control

Item Number 

Description

Category

Condition

☐ Poor

☐ Fair

☐ Good

☒ Excellent

Display 1 Data Form window

Data Table Features

Data Table displays data in a spreadsheet-like format. You can display a grid between your cells and labels for column headings. Displaying labels was highly requested for the FSVIEW procedure and is now available with Data Table.

You can control such run-time features as the ability to resize columns and to move columns using drag and drop. You can use power scrolling when you need to move across many columns. You can also print the current display or the entire table.

	First name	Gender	Age in years	Height in inches	Weight in pounds
1	Alice	F	13	56.5	84
2	Becka	F	13	65.3	98
3	Gail	F	14	64.3	90
4	Karen	F	12	56.3	77
5	Kathy	F	12	59.8	84.5
6	Mary	F	15	66.5	112
7	Sandy	F	11	51.3	50.5
8	Sharon	F	15	62.5	112.5
9	Tammy	F	14	62.8	102.5
10	Alfred	M	14	69	112.5

Display 2 Data Table window

Model-Viewer Concept

Both the Data Form and Data Table classes function through the use of a model and a viewer. The model retrieves data and delivers it to the viewer for display. You can think of the model as communicating with the data that is being displayed, whether it is retrieving the data or updating it. The model SCL is what is executed when a column is modified.

The viewer controls the display and how we interact with it. For example, the viewer controls movement within the data form or data table whether it is between rows, across columns, or between pages.

The Data Form and Data Table classes are already packaged for you with a model automatically attached to a viewer. Because the

model is automatically attached, you can call both the model methods and the viewer methods on the data form or data table. You do not have to obtain the identifier of the model to send a model method to the data form or data table; you can send it directly to the data form or data table.

DATAFORM Entry

The DATAFORM entry can be used with both the Data Form and Data Table classes to store column customizations. This is much like the modified screen for the FSEDIT procedure or the formula entry for the PSVIEW procedure.

Column customizations that can be saved include colors, fonts, capitalization, required, protected, and justification. You can also save maximum, minimum, and initial values for your columns.

For a data table, the DATAFORM entry saves changes such as column order.

For a data form, the DATAFORM entry saves your custom layout if you move your columns within a page or from page to page. It saves which classes to use to represent your columns. It also saves the location of any extra widgets you may add to your data form region such as push buttons or control objects.

The same DATAFORM entry can be shared by both a data form and a data table. This allows each object to share customizations from one catalog entry.

To specify a DATAFORM entry, enter its name in the DATAFORM Entry field of the Data Form or Data Table Attributes window. In the display below, the DATAFORM entry is called DFINV.DATAFORM and it is in the current catalog.

Display 3 Data Form Attributes window

Once a DATAFORM entry is already created, you can either leave its name in the object attributes window for your data form or data table, or you can remove its name and associate it programmatically using the `_SET_ENTRY_` method. This is usually only done if you also use the `_SET_DATASET_` method to programmatically associate your table (data set).

If you have a table and a DATAFORM entry specified in the object attributes for a data form or a data table and you want to remove the table name and associate it programmatically using the `_SET_DATASET_` method, be careful that your DATAFORM entry is not overwritten. If you remove the table name but leave the

DATAFORM entry specified, the DATAFORM entry is updated when you leave the object attributes window and will lose any column customizations. This is because the DATAFORM entry stores customizations for the currently named table. If no table has been named, then the DATAFORM entry stores customizations for a 0-column data set.

Model SCL Entry

You can also specify the name of a model SCL entry in the Data Form or Data Table Attributes window. In display 3, the name of the model SCL entry specified is DFINV.SCL.

The model SCL entry is separate from the frame's SCL entry. A model SCL entry is associated with a particular data form or data table and its compiler knows information about the columns in the data set named in the Table field for your data form or data table. In the model SCL, you can refer to the columns by name in SCL statements without having to use methods such as `_GET_COLUMN_TEXT_` to get their values. You can also have a labeled section that corresponds to the name of the column. Statements in this label execute when the column has been modified. Model SCL is typically used to initialize computed columns (ones not linked to variables in the data set), perform error checking, and data validation.

There are four SCL system variables that you can use in model SCL code:

- `_SELF_`
- `_VIEWER_`
- `_MSG_`
- `_COLUMN_NAME_`

`_SELF_` contains the object identifier of the model that is attached to the viewer and is used with model methods. Model methods include those in the "Data Form and Data Table Methods: Model Methods" section of the Data Form and Data Table Classes chapter of *SAS/AF Software: FRAME Class Dictionary, Version 6, Second Edition* (which appears in the online documentation under the Help pull-down menu in Release 6.12). Model methods are delegated to the Data Set Data Model and Data

Set Model Classes. Use `_SELF_` for the object-id parameter in those methods.

`_VIEWER_` contains the object identifier of the viewer and is used with viewer methods. Viewer methods include those in the "Data Form Viewer Methods" and "Data Table Viewer Methods" sections of *SAS/AF Software: FRAME Class Dictionary, Version 6, Second Edition*. Use `_VIEWER_` for the object-id parameter in those methods. Note: you can also use `_VIEWER_` with model methods.

`_VIEWER_` will forward those methods down to the model to be processed.

`_MSG_` contains the text to be displayed on the window's message line the next time the window is refreshed.

`_COLUMN_NAME_` contains the name of the column for which the SCL column label is being executed. If multiple columns share the same section of model SCL code,

`_COLUMN_NAME_` contains the name of the column currently causing the code to execute. For example, the following section of model SCL code executes once when ITEMNUM is modified and again when CATEGORY is modified. The value for `_COLUMN_NAME_` corresponds to which column has been modified.

```
ITEMNUM:
CATEGORY:
  put _column_name_ = ;
  return;
```

`_COLUMN_NAME_` will be blank and should not be used in any of the reserved labels such as DFINIT, INIT, MAIN, TERM, and DFTERM. `_COLUMN_NAME_` is only valid in model SCL.

You can access your model SCL from the Form
→ Edit SCL... or Table → Edit SCL...

items from the build-mode pop-up menu of your data form or data table. Once you have specified the name of an SCL entry in the Data Form or Data Table Attributes window, the Edit SCL... item is ungrayed from the pop-up menu.

You must compile your model SCL entry from within the frame, not from the BUILD directory window. Either issue the COMPILE command

from the EDIT SCL window, or select the Form
→ Compile SCL or Table → Compile SCL
items from the build-mode pop-up menu.

Creating a Data Form

When you create a data form, you can either accept the default layout of the columns and the widgets used to display them, or you can make changes. You must have a DATAFORM entry specified for the changes to be retained. By default, the data form will be drawn for you using input field widgets for the data portions and input field label widgets for the label portions of the columns. The widgets for the columns are arranged vertically. If there is room for another set of label and data widgets for more columns to the right of the current widgets, then another vertical set of widgets will be created on the page.

Before the data form is initially drawn, you can limit which columns are displayed by selecting

Customize Columns in the Data Form

Attributes window and moving any columns you do not want to display to the Nondisplayed list box. You can also make other customizations that will be used when the data form is created. You can select the Classes button and change the class used for a particular column to something other than the input field.

Once you end from the Data Form Attributes window, the data form region is populated with the columns you specified using any customizations you set. If there are more columns than will fit on one page, more pages are created for you and earmarks are automatically placed in either the upper left, upper right, or both corners of each page. These earmarks make the layout look as though it has a turned down page. You can click on these earmarks to go back and forth between the pages. Display 4 shows a data form using the default layout.

ITEMNUM:	B3200
DESC:	Diamond Tennis Bracelet
CATEGORY:	Jewelry
LOCATION:	L2QUAD4
DATEPUR:	10SEP97

Display 4 Data Form with Default Layout

You may want to display your columns using SAS/AF FRAME widgets other than the input field widget that is used by default. You can do this in several ways. Before the data form is initially drawn, you can go into Customize Columns from the Data Form Attributes window and specify different classes to use when drawing each column. Widgets from these classes will then be used when the data form is initially drawn.

Another approach is to make a widget within your data form region and then link it to a column. You can display the Column Window using Form → Display Column Window from the data form's build-mode pop-up menu to get a listing of all the columns available. You would select the column in the Column Window and then drag it onto the widget in your data form region. When you release the mouse, you should receive a note that the column has been linked to your widget. If you drag the name of

the column onto an empty area of the data form region, then a new widget representation is created using either the default input field class or any class you specified in the Customize Columns, Classes window of the Data Form Attributes window. Display 5 shows a data form with a custom layout.

Display 5 Data Form with Custom Layout

If you have rearranged the columns within your data form region or added new widgets, these customizations are stored in the DATAFORM entry. Once you have made such customizations, if you subsequently select Form → Refill Using Attributes from the data form's build-mode pop-up menu, the data form display will be recreated and you will lose any of these customizations. Be careful not to make this selection unless you truly want to start over with your layout.

Copying Widgets

As long as a DATAFORM entry is specified, the changes you make will be saved. You can also move the widgets around within the data form region and their new locations and the columns to which they are linked will be retained.

If you want to move a widget or a group of widgets from one page of the data form layout to another, you can use several items from the data form's build-mode pop-up menu. Select the widget you wish to move and then select Cut

to Form buffer from the pop-up. Move to the new page and select Paste from Form buffer from the pop-up. The widget will be removed from the previous page and created on the current page. If you want to repeat a column and not delete the original widget from the data form, you can select Copy to Form buffer instead.

If you want to copy or move a group of widgets at one time, draw a region around the widgets you want to move and fill the region with a container box. Select the container box. Then select either Copy to Form buffer or Cut to Form buffer from the pop-up menu. Move to the desired page and select Paste from Form buffer. The widgets will be drawn as well as the container box. Select the container box and then select Empty from the pop-up menu. Now select Remove from the pop-up menu. Only the container box should be removed and the widgets should remain.

Adding Widgets to Perform an Action

You may want to add widgets such as control objects to your data form region to perform some action such as providing a selection list of values for a column. You will need some SCL code to execute to display the selection list when the control object is clicked on or modified. Even though this control object would not be linked to a column that corresponds to a variable in your data set, you would still place this code in the data form's model SCL entry. You will need to link a computed column to the control object in order to have the code in the model SCL entry execute when the control object is selected.

You can drag `Make_Computed_Column` from the data form's Column Window onto the control object. When you release the mouse, a computed column by the same name as the control object will be created for you. You will receive a note that the computed column was created and linked to your control object. You would then create a labeled section by the same name as the computed column in your data form's model SCL entry and place the code there to execute when the control object is selected.

Browse Mode

When your data form is in browse mode, all columns are protected. Although you can type into columns that are protected, the widgets revert to their previous state when you press enter. This means that any new value you entered will be removed and the previous value will be displayed. You cannot tab into or tab out of a protected column and the labeled section in model SCL for a protected column does not execute when the column is modified.

If you have a navigational button within the data form region linked to a computed column, the corresponding labeled section in the model SCL code will not execute when the button is selected while in browse mode. However, if you have command processing set in the object attributes for the widget itself, the command will still be issued while in browse mode.

Creating a Data Table

When you create a data table, data are not displayed in individual widgets as in a data form. Instead, data are displayed in individual cells within a grid-like structure that resembles a spreadsheet. Multiple rows are displayed in tabular format with as many columns visible as your region allows. You can scroll horizontally to view more columns.

You can customize the data table layout to use column labels as column label text instead of column names. You can also rearrange columns at build-time and have this, as well as other customizations, stored with the `DATAFORM` entry associated with this data table object.

Moving Columns

You can rearrange columns in a data table while in build-mode and the `DATAFORM` entry will remember the location of the columns. To move columns in build-mode, follow these steps:

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

If the column you want the selected columns to follow is not currently visible, you can move your cursor to the edge of the data table object as you drag the column. This will cause the data table object to "power scroll" to columns that aren't currently visible. When the new location is visible, drop the dragged column onto the column that you want it to follow and release the mouse button.

If you would like to set the column order programmatically instead of actually moving the columns at build time, you can use the `_SET_DISPLAYED_COLUMNS_` method instead. With this method, you list all columns you would like to display in the order you want them displayed.

You can also rearrange columns in your data table at run-time, however, the new column order is not saved in your DATAFORM entry.

Editing data

To edit data in a Data Table object, you must be in edit mode. You can then be in either record-level or member-level access. In record-level access, you must lock a row before you can edit its data. By default, you must double-click on the row to lock it. If you would like a single-click to lock a row, you can specify this in the object attributes for the data table. Select **Customize Table** → **Setup** and specify **Single Click for Lock Row On**. Record-level access would be used if you plan to have multiple people editing the data at the same time using the SAS/SHARE® product.

If you use member-level edit access, then only one process can have the data open at a time. Since we open the table when you edit a frame that contains a data table, you cannot simply issue the TESTAF command while editing the frame. You can, however, end from the frame and place a T beside the frame in the build directory window. When you edit data using data table in member-level access, you do not need to lock a row before you can edit it, although you may need to click in the table to activate a cell.

When editing data, you may receive ERROR: Cell is protected when you attempt to edit a cell. There are several items to check to determine why the cell is protected.

1. Did you specify **Edit for Data Mode** in the data table's object attributes window?
2. Did you associate the table with the data table object using the `_SET_DATASET_` method instead of specifying the name of the table in the data table's object attributes? If so, did you specify the **EDIT** option on the `_SET_DATASET_` method?
3. If you are in record-level access, did you lock the row with a double-click?
4. Is the data value truncated? If there is not room for the entire data value to display in the cell, the value is truncated and the cell is then protected. See the following

discussion on how column width is determined.

Determining Column Width

Column width in data table is determined by multiplying the formatted length of the column by the width of either a zero (0) for values containing only numbers, or a lower-case "n" from either the label or default data font, whichever is wider.

It is also important to realize that the default data and label fonts for the data table object are proportional fonts. Consequently, when you display values with numbers or letters that are wider than a zero or an "n", data may appear truncated within a cell.

For example, if you have a date variable to which you have assigned the DATE7. format, the width of the column will accommodate 7 "n"s. Because the DATE7. format uses upper-case letters to represent the month, date values for this column may appear truncated.

The width can be changed interactively. In build-mode, you can resize the column using your mouse and the new size will be stored in the DATAFORM entry. At run-time, you can also resize the column using your mouse but the new size will not be stored.

To set the width programmatically, call the `_SET_COLUMN_ATTRIBUTE_` or `_SET_COLUMN_ATTRIBUTES_` method and specify the `COLUMN_WIDTH` argument. In Release 6.12 of the SAS® System, you would also need to override the `_CLEAR_SOFT_RESIZES_` method. For more details, refer to SAS Note D080.

Setting Column Customizations Programmatically

While you can customize columns under the **Customize Columns** section of the object attributes for the data table object, it seems that many data table users like the flexibility of specifying their data set on-the-fly using the `_SET_DATASET_` method in the frame entry's

SCL program. Since the data set isn't specified in the object attributes for the data table, you cannot save customizations with the DATAFORM entry for the individual columns that are not available. You can, however, specify column customizations programmatically once the data set is associated with the data table using the `_SET_COLUMN_ATTRIBUTE_` or `_SET_COLUMN_ATTRIBUTES_` method. The `_SET_COLUMN_ATTRIBUTE_` method lets you set a single column attribute and the `_SET_COLUMN_ATTRIBUTES_` method lets you set multiple attributes for a single column with an SCL list.

Common attributes set using these methods include:

- **LABEL** for custom labels that may be different from the label stored with the variable in the data set
- **FORMAT** or **INFORMAT** to allow you to specify something other than what is stored in the data set
- **LABFONT** or **DATFONT** to specify a custom font for a certain column.
If you want to specify a certain font for the labels for all columns or the data for all columns, you can use the `_SET_LABEL_FONT_` or `_SET_DATA_FONT_` methods instead.

Traffic Lighting

A common request is to be able to color-code ranges of values within a column in a data table. This is often called "traffic lighting." A value from 0 to 5 could be red, from 5 to 10 could be yellow, and greater than 10 could be green. A new method in Release 6.12 of the SAS System makes this easy to accomplish. In the INIT section of the model SCL entry associated with your data table, you can use the `_SET_VIEWER_ATTRIBUTE_` method. The INIT section executes as each row is read in so the `_SET_VIEWER_ATTRIBUTE_` method can conditionally set the color for that row in the column based on the values you provide. If the column values can be updated, you will also need to call this method in the column's labeled section in the model SCL entry.

Held Columns

Users often want columns to remain stationary while they scroll the table horizontally to see other columns. The columns that remain stationary are called held columns. This can easily be accomplished by using the `_SET_HELD_COLUMNS_` method. You pass a list to this method that contains the number of the first column to hold and a list that contains the number of the last column to hold. These columns and those in between will then remain stationary as you scroll the table horizontally. If you only want to hold one column, you can use the same list for both the first and last columns to hold. For an example of using this method, refer to the INPUT/OUTPUT item by Lynn Leone on page 71 of *OBSERVATIONS®: The Technical Journal for SAS® Software Users, Vol.6 No. 2*.

Conclusion

The Data Form and Data Table objects are useful for viewing and editing data in a graphical manner. By storing customizations in a DATAFORM entry and using model SCL code, you can control the display and perform data validation for both objects. For more information on the methods discussed in this paper, refer to the "Data Form and Data Table Classes" chapter in *SAS/AF Software: FRAME Class Dictionary, Version 6, Second Edition* in the online documentation for Release 6.12 of the SAS system.

For more examples and discussions of the Data Form and Data Table objects, refer to the following articles in *OBSERVATIONS: The Technical Journal for SAS Software Users*:

- *Introducing a GUI Approach to Data Entry Using Data Table and Data Form Objects in SAS/AF FRAME Software* by Ann Carpenter and Lynn Leone, pages 6 - 14, Vol. 5 No. 2.
- *Creating a Contacts Application Using New Classes in SAS/AF Software* by Patti Brideson, pages 35 - 42, Vol. 5 No. 2.

- *Data Validation Using the Data Form Object in SAS/AF FRAME* by Ann Carpenter, pages 6 - 14, Vol. 6 No. 1.
- *Data Table and Data Form Enhancements in Release 6.12* by Scott Wilkins, pages 28 - 35, Vol. 6 No. 2.

There are also examples for the Data Form and Data Table objects in *Solutions@Work: Rapid Applications Development Examples*, available from our Publications department.

Author's Note: Information from the "Data Form and Data Table Classes" chapter of the

SAS/AF Software: FRAME Class Dictionary, Version 6, Second Edition in the online documentation for Release 6.12 of the SAS system and information previously written by Ann Carpenter Rockett and Lynn Leone contributed to the material in this paper.

SAS, SAS/AF, SAS/FSP, SAS/SHARE, and OBSERVATIONS are registered trademarks of SAS Institute Inc. in the USA and other countries. @indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.