Using SAS® Views for Data Source Visualization

Steven A. Wilson, Resource Biometrics, Inc., Emeryville, CA.

Abstract
SAS Views are extremely powerful tools for managing data files with the SAS System. SAS views are compiled SAS statements that define or describe data that are physically stored somewhere else. The physical data referenced by a SAS view may be a SAS data set or some other file structure that is accessible via SAS. Once created these views are used like any regular SAS data set.

In an environment with many different types of frequently accessed data sources, it can be useful to have a shared set of SAS views that are used by a large number of personnel. These views can be used to examine the many different data sources in a similar format, using consistent SAS variable names, labels, and formats.

After a brief discussion of the pros and cons of using SAS views, this paper will discuss a method of using SAS views in conjunction with SAS/AF extended tables for displaying the contents of a data source. The emphasis of this paper will not be SAS/AF development, but instead will focus on the use of SAS views to help visualize a data environment consisting of many different data sources.

Introduction
This paper is a case study in the use of SAS views to provide meta-data about available data sources in a large, multi-application environment. In this case, a system was desired to allow personnel at Kaiser-Permanente’s Division of Research to easily examine the contents of various data sources. These data sources reside in different database management systems (DBMS), including SAS data sets.

The system under review, named DAD, was developed to accomplish two specific goals:

1) Allow a casual user to easily browse the contents of a data source, regardless of how the data are stored. Achieving this goal is the subject of this paper.

2) Enter specifications for extracting data via generated SAS code. This easy access to data eliminates the burden of extracting data from unfamiliar or difficult to access data sources, allowing researchers to concentrate on analyzing data.

DAD Functional Specifications
Listed below are the major components of the functional requirements for the DAD System. The specifications indicated by @ are those specifications that will impact the type of SAS view that will be used.

• Full-screen interactivity
  - custom selection lists
  - on-line help
  - quick response time

• Easily allow new data sources
  - new data sources will be regularly added to the computing environment

• Handle multiple types of data sources
  - SAS data sets on direct access devices
  - Sequential SAS data sets
  - Flat files on tape or direct access
  - DB2 tables
  - Possibly other types of data sources

• Examine contents of data source
  - Variables and attributes
  - Provide consistent variable names, labels
  - Attach appropriate formats to variables

• Provide derived data elements
  - Provide appropriate computed variables
  - May need to join tables to display useful information about the data source
Multiple views of a single source
- Some data sources, especially those involving joined tables, may result in multiple views, where each view represents a different way of organizing and presenting the data.

Easy to maintain
- Do not want this to be a high maintenance system

• Specify extract criteria & generate code to extract data
  - Allow specification of WHERE conditions
  - Provide selection list of discrete values
  - Whoa! This discussion will have to be in another paper!

Since most of these specifications will be determining factors in the type of SAS view that is most appropriate for a particular data source, let's take the next few pages to examine the types of SAS views available.

SAS Views
There are many different types of data views in the SAS System. Regardless of the type of SAS view used, a view does not contain any data values. Instead, views contain machine code that defines data or describes data that are stored in another place.

The main advantages of using SAS views include:

• Views of complex data structures can be created for easy use.
• Views can be used to limit access to data in a master data set.
• Storage space is saved since only the view definition is saved instead of a whole data set.
• The input data is always current because data from views are derived at execution time.

Below is an outline listing the various types of views available in the SAS System. The type of view used should depend upon the needs, complexity, and uses of the view.

View of SAS files
- SQL view
- DATA step view

View of non-SAS files
- DATA step view
- SQL Pass-Through view *
- SAS/ACCESS view *

(* requires necessary SAS/ACCESS interface)

Once created, use the view like a regular SAS data set. However, since a view does not contain any data, data cannot be modified in a view (except in the special case of SAS/ACCESS views).

Views of SAS files
When working only with SAS data sets, you may create views with either a DATA step or PROC SQL. SQL views, while requiring knowledge of PROC SQL, are generally more efficient. This is because of PROC SQL's powerful data manipulation capabilities and the SQL query optimizer, which when passed multiple and/or layered views (views that reference other views), will optimize the entire set of compiled SQL queries together.

The code below creates a simple SQL view that is referenced by the name SASUSER.MALESCAL.
Notice that this view defines a variable that does not exist in the input data set.

```/* Create an SQL VIEW */
PROC SQL;
CREATE VIEW SASUSER.MALESCAL
AS
SELECT *
, ((IDATE-DOB)/365.25) AS AGE
LABEL = 'Age at Index Date'
FORMAT = 3.0
FROM LIB.SOCAL
WHERE SEX = 'M'
QUIT;
```
A more complicated, and less efficient, SQL view appears below. This view concatenates two data sets together and calculates a new variable that does not exist on either of the input data sets.

```sql
/* Create an SQL VIEW */
PROC SQL;
CREATE VIEW SASUSER.MALE AS SELECT *
  (IDATE-DOB)/365.25) AS AGE
  LABEL = 'Age at Index Date'
  FORMAT = 3.0
  FROM (SELECT *
    FROM LIB.SOCAL
    OUTER UNION CORR
    SELECT *
    FROM LIB.NORCAL)
  WHERE SEX = 'M' ;
QUIT;
```

The other type of view that can be made of a SAS data set, DATA step views, are the most flexible because of the availability of DATA step programming statements. Usually, though, this type of view is more costly than SQL views.

To create a DATA step view, specify the VIEW= option on the DATA statement. Note that the view name must appear twice in the DATA statement as shown below.

```sas
/* Create a DATA step VIEW */
/* VIEW name must appear twice*/
DATA SASUSER.MALE / VIEW=SASUSER.MALE ;
SET LIB.SOCAL (WHERE=(SEX='M')) ;
LIB.NORCAL (WHERE=(SEX='M')) ;
AGE = INT((TODAY() - DOB)/365.25) ;
RUN ;
```

**Views of Non-SAS files**

DATA step views can also be created for external files. Once the view of the external file or DBMS database is created, SAS programs may reference the view, eliminating the need to know the layout of the external file or DBMS database. Because of the number of varied research databases at the Division of Research, this is the holy grail that leads to the development of the DAD System.

By utilizing the DATA step view, you can create a view of any type of file that can be read by a DATA step.

The following DATA step creates a view of an external file. When data is needed from this file in future jobs, it can simply be referenced by the SAS data view name SASLIB.HOSP. Notice that this view calculates the AGE variable as of TODAY(), thus the computed age is always computed as the age at the time the data is read by the view.

```sas
DATA SASLIB.HOSP /
  VIEW=SASLIB.HOSP;
  INFILE 'DOR.HOSP.YRS7992';
  INPUT @1 ID $7.
  @8 GENDER $1.
  @9 DOB MMDDYY6.
  .....................
  ;
  AGE = (TODAY() - DOB)/365.25;
RUN;
```

Other types of SAS views that reference non-SAS data files can be created using the SAS/ACCESS product. The types of SAS views available with SAS/ACCESS are described after a short description of the SAS/ACCESS product.

**SAS/ACCESS**

SAS/ACCESS is the SAS System product that allows your SAS session to seamlessly access data that is stored in non-SAS data structures. Each type of data source you wish to access would require a separate purchase of a SAS/ACCESS product.
interface because each SAS/ACCESS interface can access only one type of data source.

SAS/ACCESS allows two methods for extracting data in a DBMS table - SAS/ACCESS views and the SQL Pass-Through Facility. For each method, you should strive to extract only the desired observations and variables from the DBMS file. Extracting or processing non-essential data only increases costs and lowers efficiency.

Each of these methods to extract data from a DBMS table is described below. Be advised, however, that both methods are not available for all types of data sources. For example, SAS/ACCESS views are not supported by the SAS/ACCESS interface to ODBC.

**SAS/ACCESS Views**

PROC ACCESS, which is usually executed as an interactive windowing procedure, allows you to quickly and easily create a SAS view of a DBMS table. This is done by making selections from a number of ACCESS procedure windows to create descriptor files. These SAS/ACCESS descriptor files provide the mapping between a DBMS table structure and a SAS data set.

This process may also be performed via the Access Window of Display Manager or by submitting PROC ACCESS batch code. PROC ACCESS is usually run interactively, but SAS Technical Report P-221 describes the batch syntax.

There are two types of descriptor files that must be created to define a SAS/ACCESS view of a DBMS table - the Access Descriptor and the View Descriptor. Both of these descriptor files must be created prior to accessing an existing DBMS table from SAS.

Access Descriptors are the master description of a single DBMS table. The Access Descriptor file is created first and is used to identify the DBMS table being accessed and select columns from the table. The Access Descriptor will contain default SAS variable names, labels, and formats, but you may assign your own SAS variable attributes.

From an Access Descriptor any number of View Descriptors may be created. These views select all or some of the data described in a single Access Descriptor and may also contain assigned SAS variable names, labels, and formats.

Once created, the View Descriptor may be used like any other SAS view, or, in other words, like any SAS data set in a SAS program. These views look and behave just like regular SAS data views.

The code below creates both an Access Descriptor and a View Descriptor for a DB2 table and then uses the view for subsequent processing.

```
PROC ACCESS DBMS=DB2;
  CREATE SASUSER.DB2.ACCESS;
  SSID=DSN2;
  TABLE=PAT.VPATIENT;
  ASSIGN=YES;
  LIST ALL;
  CREATE SASUSER.DB2.VIEW;
  SELECT ID NAME DATE SEX FLAG;
  SUBSET WHERE FLAG = '';
RUN;
```

```
PROC PRINT DATA=SASUSER.DB2(where=(SEX='M'));
RUN;
```

```
PROC FSEDIT DATA=SASUSER.DB2;
RUN;
```

```
/* Be careful ! */
DATA JOINED;
MERGE SASUSER.DB2
  WORK.TWO_OBS (IN=OK);
  BY ID;
  IF OK;
RUN;
```
Notice that unlike the previous types of views we have seen, ACCESS views do not allow for the creation of new fields, or the manipulation of more than one table. With ACCESS views, you can only create a view of the existing columns in a single DBMS table.

An important feature of ACCESS views is that all WHERE conditions specified are optimized and passed to the DBMS. In the above code, only the DBMS table rows meeting both the specified WHERE conditions (FLAG='*' and SEX='M') are passed to PROC PRINT for processing.

Also note that the DBMS table can be directly edited using PROC FSEDIT or a similar method. This is a feature that is available only with ACCESS views. While certainly useful, this functionality is not necessary for the DAD System.

It is important that you exercise caution when using SAS/ACCESS descriptor files to access large DBMS tables. This is because all data contained in a DBMS table referenced by the ACCESS view is passed to SAS unless a WHERE allows the DBMS to perform some subsetting. Merging any file with an ACCESS view of a large DBMS table is, in most cases, prohibitively expensive.

Another problem with using SAS/ACCESS descriptor files is that if the structure of the DBMS table changes, all Access and View descriptors created for the DBMS table will become useless and have to be recreated. No part of the descriptor files may be salvaged when the structure of the underlying DBMS table changes.

When an ACCESS view references a data source that frequently changes, this is a big problem. The problem is intensified if you have invested a lot of time to create the descriptor files by assigning SAS variable names, labels, and formats using interactive windows.

PROC SQL Pass-Through Facility
With the SAS/ACCESS product there is another method for accessing DBMS tables from a SAS session. The Pass-Through facility of the SQL procedure allows you to pass SQL code directly to the DBMS system. Thus, the target DBMS evaluates and executes SQL code that is passed-through, passing back to SAS only the results of the evaluated passed-through code. This can be extremely efficient because the DBMS data can be evaluated in its natural state, realizing the inherent efficiencies of the DBMS.

Below is an example showing a sample query using the SQL Pass-Through Facility to retrieve data from a DB2 table.

```
PROC SQL;
/* CONNECT TO makes DBMS connection */
CONNECT TO DB2 (SSID = DSN2 );
CREATE VIEW SASUSER.MALE
AS
SELECT */
/* CONNECTION TO extracts from DBMS */
FROM CONNECTION TO DB2
/* This "subquery" is processed by */
/* the DBMS. Only the results are */
/* passed back to SAS. */
SELECT */
/* DBMS table and alias */
FROM ADT. VMES IUS DATA ALIAS
WHERE CODE BETWEEN '100' AND '400'
/* End of passed-thru code */
WHERE PUT (CODE, $OK. ) = 'OK' ;
QUIT ;
```


With the PROC SQL Pass-Through facility it is important to make the distinction between the SAS SQL code and the passed SQL code. The passed SQL code (WHERE CODE BETWEEN '100' AND '400') will be evaluated by the target DBMS and may contain syntax specific to the DBMS's implementation of SQL. However, the SQL code that is not passed-through (WHERE PUT (CODE, $OK. ) = 'OK') is
evaluated by SAS and must conform to the SAS System's implementation of SQL.

This illustrates the one important rule to remember when working with SAS/ACCESS: "Move data selection and manipulation logic as close as possible to the data source" (Wallace, 1994). This will help ensure that this tool works most efficiently by passing to SAS only the data needed. In other words, you are well advised to do as much subsetting as possible in the passed-through code or WHERE conditions to make your views most efficient.

SQL Pass-Through code, like all the previously shown code, may be submitted for execution via a batch job, an interactive SAS session, or as remote submitted code via a SAS/CONNECT session.

Applications Development

Now I know enough about SAS views to develop the DAD System. Remember, code that defines a SAS data view does not execute. It is simply compiled and stored for future use. Thus, you can think of a view as a "recipe" that must be executed each time the view is referenced.

Be warned, though. Each time a view is referenced it must execute the underlying compiled SAS code to create the data before performing any requested actions. So, for example, running a PROC PRINT on a SAS data view is like executing a DATA step (or SQL Query, depending upon the type of view) followed by a PROC PRINT.

Since this overhead is associated with a SAS view each time it is referenced, SAS data views are very costly if frequently used. If a view is going to be referenced more than once during a SAS session, you are well advised to create a SAS data set that contains the data described by the view and process the data set rather than the view.

In the DAD System, however, the developed views are not used for extracting data. Only the data structure, or contents, the view represents is being used. This contents is easily obtained from the view without the high overhead associated with extracting data using a view.

Finally, it should be noted that when a view is created, you should be sure to keep the source code. Source code cannot be obtained from the compiled version of the view. Without the source code you will be unable to determine how the view was created or what it describes. Don’t lose the source code!

See Norton (1994) for the definitive SUGI paper concerning SAS views.

Which Type of View & Why

Now that the different types of SAS views have been described, we need to determine which type of view will be best suited for displaying the contents of our data sources.

Since there are both SAS and non-SAS data sources, it is apparent that different types of views will be used. Listed below are the parts of the functional specifications that we earlier ascertained would be factors in determining the type of view to be used.

- Easily allow new data sources
  All types of SAS views are easily created, but if many different types of data sources are utilized, it may be too expensive to purchase numerous SAS/ACCESS interfaces.

- Handle multiple types of data sources
  The type of view would depend upon the type of data source being read. If a data source can be read using different types of views, choose the type of view that is most efficient.

- Examine contents of data source
  The data vector, which contains the contents (variables and their attributes) of a data source, is stored as part of DATA step and ACCESS views, but must be created whenever an SQL view is referenced.
Variable names, formats and labels are easily assigned via any type of view, but these attribute assignments are best made in batch code, allowing you to easily recreate the view.

- **Provide derived data elements**
  Only SQL and DATA step views allow you to perform multi-table operations and create new variables. Only the Pass-Through facility allows you to instruct a DBMS to manipulate tables.

- **Multiple views of a single source**
  There is no limit to the number of views of a data source that can be made. Thus, all types of views meet this condition.

- **Easy to maintain**
  ACCESS views are by far the easiest to create when done interactively using the Access Window. SQL views, on the other hand, can require significant coding. However, when an SQL or DATA step view is coded, the code can be saved to modify and recreate the view at a later date. When a data source changes, the contents shown by DATA step or ACCESS views would not reflect the change because the data vector is stored as part of these types of views. These types of views would need to be recreated to show the correct contents or to access the data. SQL views, on the other hand, create the data vector when accessed and will frequently work when changes to the data source are made.

If these views were going to be used for reading the data from a data source, then the WHERE optimization features of some of the views would be important. WHERE conditions that are used when referencing ACCESS views are sent to the DBMS for evaluation if possible. With Pass-Through, however, only the passed-through code is evaluated by the DBMS, meaning that any subsequent WHERE would be evaluated by SAS.

Generally speaking, when extracting data, you may wish to create ACCESS views to permit general querying of small DBMS tables (using a strict WHERE condition) and use SQL Pass-Through when very specific extracts from DBMS tables can be programmed or if your DBMS tables are subject to frequent change.

In the case at hand, no data is being extracted with these views. Since these views are not intended for use in extracting data from the data sources, we are less concerned with efficiency.

**Which Platform & Why**
In a truly distributed environment, there would be data sources residing on multiple machines. In such an environment, the DAD System would need views of sources on different platforms and the ability to move information between machines. This functionality is available through the SAS/CONNECT product.

However, managerial decisions at the Division of Research dictated that the DAD System simply provide views of mainframe data sources at the present time. Thus, the DAD System was developed on the mainframe platform.

**User Interface - SAS/AF Extended Tables**
As previously mentioned, the DAD System is an on-line, interactive system. This system must supply users with custom selection lists that display the available data sources for a user to select. These selection lists would also be used to drill-down into the selected data sources and even into specific variables within the data source. Thus, a SAS/AF application was developed to provide this desired functionality.

Because of the limited Graphical User Interface (GUI) on the mainframe platform, these extended tables were not developed using FRAME entries in SAS/AF. Instead, these selection lists are displayed using extended table program entries. The data source selection list is populated with data from a data set that identifies and describes available data sources.
When a data source is selected, the data vector of the view of the data source can be displayed in another extended table. And within the listing of data source variables, the user has the ability to drill-down into selected variables, accessing a list of values that are coded for the specified variable.

The DAD System

The query specification menu of the DAD System appears below. Recall that this system is used to generate SAS source code for extracting data according to entered query specifications. Thus, this menu contains many selections that are not of interest to us in context of this paper. We’re interested in the use of SAS views to populate the screens displayed for step 3, “Specify Data to Extract”.

Some of the listed data sources are SAS data sets, some are flat files, while others are DB2 tables. The user, however, does not need to know which is which because SAS has each listed source mapped via a view. One major goal of the DAD System has been achieved: The user does not need to know the source file layout or how to access the data source.

Some data sources may have multiple views listed, where each view arranges the data differently. Some listed sources may not even physically exist, but exist logically as a view of manipulated data for easy interpretation and use. So, care must be taken when deciding what views to create when providing a list of “data sources” to view.

This list of data sources is maintained as a SAS data set that contains information regarding the data sources. Since not all the variables in the data source identification data set are meaningful to the end user, only a subset of the variables in this data set are displayed - the data source description and the dates the data source are in effect. The displayed date range is, of course, a calculated field because the sources have moving windows of data availability (eg. The previous 27 months, excluding the current month).

Some of the other variables maintained in the data source index data set are the following:
- ID and DATE index variables
- Full name of data source file
- Type of data source

Each of the displayed data sources represents a SAS view. The type of SAS view used to describe a data source depends upon how best to read the particular data source. Regardless, code to create each of the views is very similar to the code that has been previously shown.

In most cases, the created views are SQL and SQL Pass-Through views. In the case of the data sources that are sequential files, DATA step views were used.

---

Data Source Visualization - Data Sources

When a user requests to specify data to extract, a list of the available data sources is displayed using a SAS/AF extended table program entry. This extended table is shown below:

Some of the other variables maintained in the data source index data set are the following:
- ID and DATE index variables
- Full name of data source file
- Type of data source

Each of the displayed data sources represents a SAS view. The type of SAS view used to describe a data source depends upon how best to read the particular data source. Regardless, code to create each of the views is very similar to the code that has been previously shown.

In most cases, the created views are SQL and SQL Pass-Through views. In the case of the data sources that are sequential files, DATA step views were used.
**Data Source Visualization - Variables**

If a user selects a data source to examine, the DAD system will run a PROC CONTENTS on the data source view. Regardless of the type of view or its complexity, obtaining a PROC CONTENTS of a SAS view is quick and efficient.

Another extended table entry is used to display the contents of the data source view. This display for a data source is shown below:

<table>
<thead>
<tr>
<th>Select desired variables</th>
<th>Description</th>
<th>All Variables</th>
<th>Field Buoyt</th>
<th>All Events</th>
<th>Last Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 = Variable description</td>
<td>All Variables</td>
<td>Field Buoyt</td>
<td>All Events</td>
<td>Last Event</td>
<td></td>
</tr>
<tr>
<td>2 = Select variable</td>
<td>All Variables</td>
<td>Field Buoyt</td>
<td>All Events</td>
<td>Last Event</td>
<td></td>
</tr>
<tr>
<td>3 = Derivative variable</td>
<td>All Derivative</td>
<td>Field Buoyt</td>
<td>All Events</td>
<td>Last Event</td>
<td></td>
</tr>
<tr>
<td>4 = Select specific value</td>
<td>All Derivative</td>
<td>Field Buoyt</td>
<td>All Events</td>
<td>Last Event</td>
<td></td>
</tr>
</tbody>
</table>

Recall that the variables displayed are those of the view, and not necessarily the actual physical columns of the data source. The view may create derived variables or merge the data source with another data source to obtain a rich set of variables for display in the data source variable screen.

The order in which the variables are displayed is, by default, that of the view’s data vector, which can be defined in SQL and DATA step views.

Variable names and labels have not been assigned in this view. An important aspect of the DAD System, however, is to provide a consistent set of variable names and labels, and the use of a defined set of formats for decoding the variable values.

On this variable selection list, a special selection code of “W” can be used to drill-down to view the variable’s values for WHERE specification.

**Data Source Visualization - Values**

Since the purpose of the DAD system is to generate code for performing extracts from the data sources, it is helpful to be able to display lists of valid variable values for selection when specifying subsetting criteria.

A variable value list could be maintained as:
- a SAS format
- a data dictionary table
- frequency distribution of the variable that is generated upon request (for small data sources)
- frequency distribution that is generated periodically and stored for easy retrieval when requested (for large data sources)
- a view that provides a listing of the variable values

By placing this information into an extended table, a selection list can be provided for WHERE subsetting, or simply to view the variable values.

**Generated Code to Extract Data**

Recall that ultimately, the DAD System was designed to be a code generator. While this aspect of the system is not relevant to this paper, it should be mentioned that the SAS views used for displaying the data source contents are not used for extracting data from the data sources. These views would be much too expensive to use for extracting data from large data sources. The code generated to extract data uses a generated SQL pass-through query to extract the desired data from non-SAS data sources.
The final result is that researchers and programmers now completely removed from needing to know the details of extracting data from multiple different data sources. Extracts can be easily performed regardless of how the data source is stored. Accessing data is no longer a burden, and time may be more productively spent analyzing data.

**Conclusion**
As an application developer, the SAS System gives me superior technology for use in developing state-of-the-art applications. The functionality offered by SAS views lets me offer a great variety of “pre-processed” data sources to our staff.

Database access capabilities are important too, because SAS is usually not the primary application used for data warehousing. Nor can all staff effectively use SAS. Thus, the application interconnectivity offered by SAS/ACCESS is vital for effective applications development in SAS.

The SAS System provides an integrated, flexible programming environment that allows us to optimize our resources.

**References**


Steve Wilson  
Resource Biometrics, Inc.  
5801 Christie Avenue  
Emeryville, California 94608-1928  
(510) 597-7217  
e-mail: wilson@resbiom.com

Any comments, questions, or details of your experiences are most welcome.

SAS, SAS/ACCESS, SAS/AF, and SAS/CONNECT are registered trademarks of the SAS Institute Inc., Cary, NC, USA.

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