Paper 122-25

Building a System for Uniform Data Access in a Research Environment

Julia L. Bienias, Rush-Presbyterian-St. Luke's Medical Center, Chicago, IL
Charles L. Owen, William E. Wecker Associates, Inc., Novato, CA
Woojeong Bang, Rush-Presbyterian-St. Luke's Medical Center, Chicago, IL

ABSTRACT

Do you work in a research or production environment in which you must produce many reports or statistical analyses from multiple data sets? Are your data structures sufficiently complex that you want to create a system that provides easy access to the data in a logical manner? Do you have users of different levels of sophistication and who have different goals? We were faced with these tasks, plus we needed to create a system whose code would be robust to changes in the underlying data warehouse. In our research group we have several ongoing studies with data stored in relational databases, flat files, and spreadsheets, and there are dozens of active research projects. We created a common user interface for enterprise-wide access to data from all of these sources for all of the studies, using SAS/AF® FRAME entries and Screen Control Language. We have added features that allow us to quickly and easily review our data, produce reports, and access the full power of the SAS System, saving users considerable time. We will describe some of the particular design issues we faced, show the system we created, and discuss future developments.

BACKGROUND

We work in a research group that studies health problems of the elderly. It is a very diverse group, with several large ongoing studies and many smaller ones in a variety of subject-matter areas (e.g., cognitive function, physical function, neuropathology, care giver support, Alzheimer's disease). Each study has its own set of data collection forms, often administered at more than one point in time for a given study participant. And, for any study, there may be a dozen or more active research projects connected with it at a given moment, each needing different subsets of the data for analysis.

We already had mechanisms for data entry (data are keyed from paper forms or they are entered during an interview using computer-assisted personal interviewing programs written in Blaise©) and data storage (flat files or Informix© relational database on a Sun UltraSparc™ 2 under the Solaris™ 2.5.1 Unix operating system). We had programs written in SAS that were used to read the data from flat ASCII files and also, more recently, from the Informix database directly using the SAS/ACCESS® SQL Procedure Pass-Through Facility®. We have a staff of professional programmers whose task it is to access data, create reports, perform statistical analyses, etc. However, the system had two main drawbacks. First, after several years, there were hundreds of programs on the system, many of which were interdependent. Such a system would be difficult for a new employee to learn, and it was easy for a seasoned employee to forget the details of a program used months earlier. Second, the researchers wanted better direct access to the data (as opposed to having to go through the programmers), but the system of programs was difficult to navigate. Thus, we wished to create a system with a common interface that could be used by anyone, saving time and programmer resources.

Considering our options, we decided the SAS/AF FRAME entries would provide the best solution to the problem. FRAME entries are an integral part of the SAS System's interactive applications development environment. The FRAME entry supports object-oriented programming and enables the creation of graphical interfaces that can exploit the advantages of the SAS System.

OUR SYSTEM DESIGN

We faced several special challenges. First, we could not modify anything currently used in production without jeopardizing the ongoing research, so we chose to build a system that was conceptually "a level above" the existing programs. That is, we made our Screen Control Language (SCL) programs access existing code as much as possible. Our application uses information supplied by the user together with the macro facility to generate SAS code that is independent of existing code. This way, we can change the "details" of the various programs without modifying the application. Doing this also prevented us from duplicating work that had already been done and had already been thoroughly tested.

Second, we wanted a system that would be as easy to maintain as to use. Specifically, we wanted maintenance to take place outside of the system. That is, if something changes or if we needed to add pieces, no recompiling or rewriting of the underlying SCL is required. All of the maintenance should be focused on the internals of the macros and a few simple text files. As an example, before we began creating the application, there were separate programs that read data from each of the many, many questionnaires and data collection forms. We created a single macro structure that can access any of the programs based on a few keywords provided by the user (via the interface). Maintenance then consists only of associating the keywords with the appropriate program. The structure we used is:

%studynm(year= , form= , keep= , dsn=),

where "form" refers to the questionnaire or data collection form, the "keep" variables are in addition to ID and other variables needed for linking data sets, and "dsn" is the output data set, which is the name of the project defined by the user (see next section). This structure is "behind the scenes;" the user simply chooses the year, form, & variables by clicking on the desired choices on the screen.

Third, we wanted the system to follow the current logic used in our organization. Previously, researchers requesting reports or data analyses from the programmers would provide the study name and variables needed, listing

them by data collection form or questionnaire. Our generalized structure maintains this organization.

The SAS System was the best choice for what we needed, because it integrates data access and analysis tools seamlessly.

OUR APPLICATION

The user first sees a screen with four options: Data, Programs, Browse, and Query (see Figure 1).

The Data Sub-Menu

Data access is organized around "projects," which is a structure that is both logical and familiar to our users. Each user can create, modify, and delete as many projects as he or she wants. The "project," which is an SCL list, stores all the information necessary to create a data set suitable for analysis (e.g., which variables are desired, whether they are to be formatted or not). These SCL lists are designed to be easily modified within the application and are used in conjunction with the macro facility to create robust, error-free code. Once created, the user does not need to re-create it, as the projects are stored from session to session as an SLIST catalog entry, which can also be modified within the application. (See Figure 2.)

When first starting the application, users typically choose the Data option first to create the data set, although the other sub-menus can also be accessed independently. From the Data option, they choose a "study" and a "project" (Figure 3). At this point they can either choose an existing project or create a new one. A list of existing projects pops up when the “Project Name” down-arrow is selected.

Once a study is selected and a project is named, the user moves to the data window, which displays all forms and variables associated with the given study (Figure 4). The researcher can then choose which variables from which data collection forms, and for which years (e.g., baseline, follow-up year 1, follow-up year 2, etc.) to be associated with the project. There can be any number of projects associated with a given study, and this information will be stored for future invocations of the application. A particularly nice feature of the system is the ability to modify or delete projects. Suppose the researcher realizes he or she forgot to include a variable to the data set; it can be added simply by clicking on it and then choosing "Generate Code" from the "Action" menu (as shown in Figure 4).

Because our studies are primarily longitudinal in nature, it was critical to have the data linked appropriately. The application generates code that automatically merges or concatenates the data as appropriate, giving the user a data set with data spanning the multiple data collection periods of the study in a form that can be directly used in analyses.

We added some additional special features needed by our research group, which we expect would be common to many organizations. First, we have many categorical variables, and we thought it would be easier to have the formats associated with them automatically. So, we created a Common Format Library to store the formats, and the labels of the formats include the original value (e.g., "1: Male", "2: Female"). Second, we have codebooks for the various data collection forms online, and we made our AF system link to them. If a user sees a variable that is unfamiliar, he or she can click on "Codebook" and there is a search feature built in that will bring up the definition of that variable on demand. (See Figure 5.) Finally, we have some special computed variables that are used frequently (for example, collapsing age categories into "65-74" and "75 and over"); we created a separate entry to allow the user to choose which, if any, computed variables to add to the data set for a given project. These computed variables vary by study, but the interface does not change. Figure 6 shows this feature; the first column is a “check box” to show which modules have been selected.

The Browse Sub-Menu

From the main menu (Figure 1), choosing “Browse” will allow the user to review the data, produce frequency tables, generate simple statistics, and do simple or complex queries; none of these tasks requires the user to know anything about SAS syntax. The tasks can be performed using the data just created from the Data Sub-Menu, or from a previously created data set. The more advanced user can proceed directly to the full SAS system by calling up the Program Editor and Log windows from the “Window” pull-down menu.

We incorporated browsing by calling up the SAS/FSP® table browse feature. In addition, because writing queries was not familiar to our users, we built a special interface for building “where” clauses (Figure 7). Choosing a “where” clause will then subset the data for all subsequent analyses, until the “where” clause is removed. A title is added to all windows and output indicating the active “where” clause. Frequency tables and univariate statistics are available as well, on the full or subsetted data set.

The Programs Sub-Menu

Choosing “Programs” from the main menu (Figure 1) allows the user to run pre-written programs. This feature can be very useful for a researcher who might wish to obtain a particular data report on-demand, for example.

The Query Sub-Menu

At this stage, we have left this sub-menu open to future development of commonly-used “canned” queries.

CONCLUSIONS

The system we built assumes the user is familiar with data but not necessarily with the structures in which the data are stored nor with SAS. This allows the end-user, whether a programmer or researcher, to focus on data analysis and reports, not on how to get the data; this results in more efficient use of programmer time and researcher time. It allows uniform and dynamic access in a setting in which data are constantly being updated.

The system we have developed could be successfully used in any setting with multiple data sources and multiple needs.

REFERENCES

SAS Institute Inc. (1989). *SAS/FSP® Software: Usage and Reference, Ver. 6, First Ed.* Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1993). *SAS/AF® Software: FRAME Entry Usage and Reference, Ver. 6, First Ed.* Cary, NC:

SAS Institute Inc.

SAS Institute Inc. (1995). *SAS/AF® Software: FRAME Application Development Concepts, Ver. 6, First Ed.* Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1997). *SAS/ACCESS® Software Changes and Enhancements, SQL Procedure Pass-Through Facility, Ver. 6.* Cary, NC: SAS Institute Inc.

CONTACT INFORMATION

Please address correspondence to:
Julia L. Bienias, Sc.D.
Rush Inst. for Healthy Aging
1645 W. Jackson Blvd., Suite 675
Chicago, IL 60612
e-mail: *jbienias@rush.edu*

ACKNOWLEDGMENT

We thank Ming Gao for creating the codebook search program.

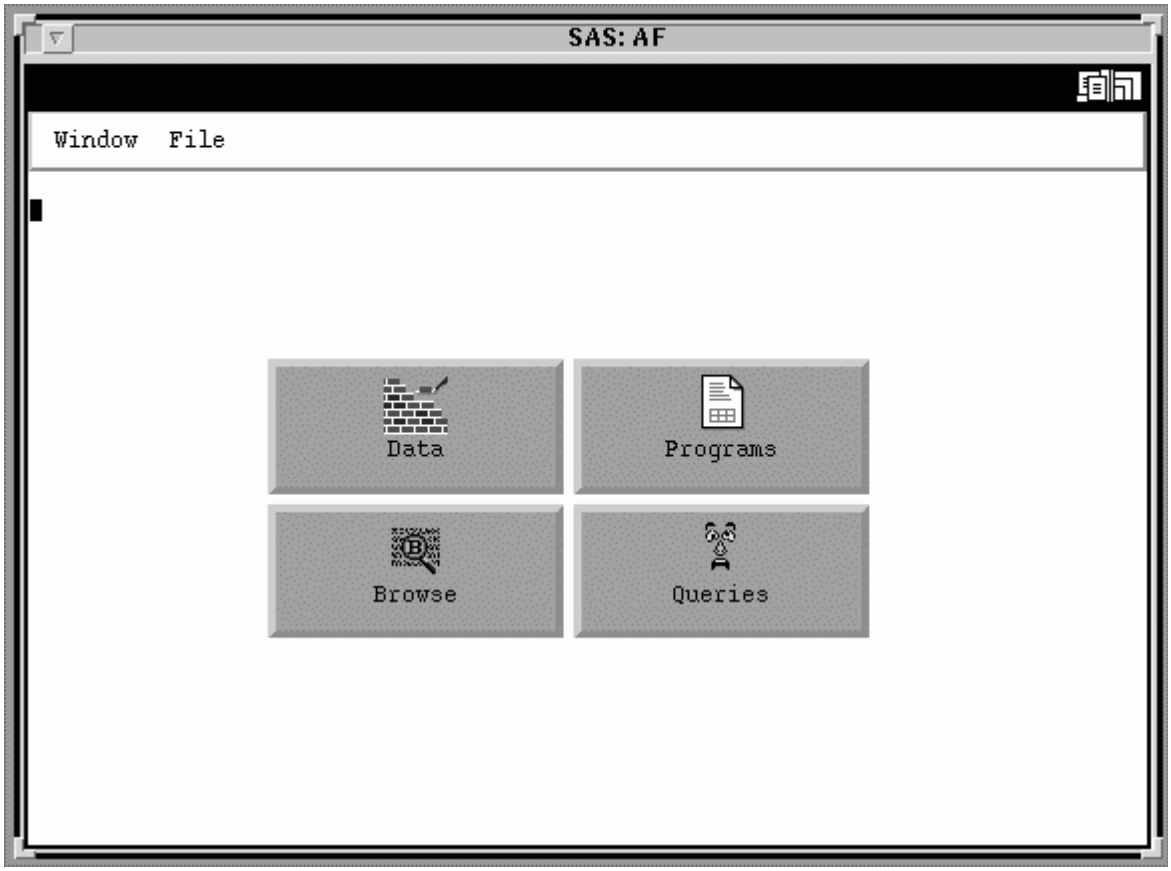

Figure 1

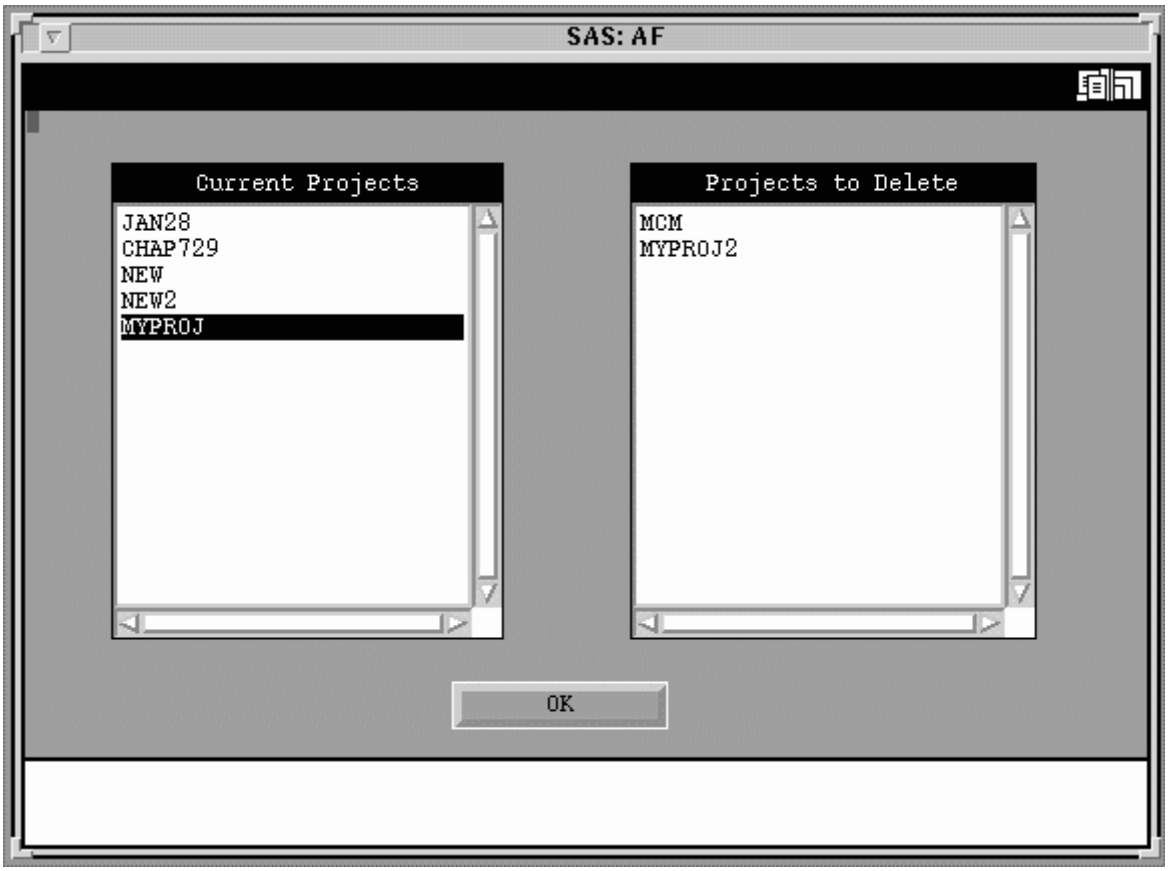

Figure 2

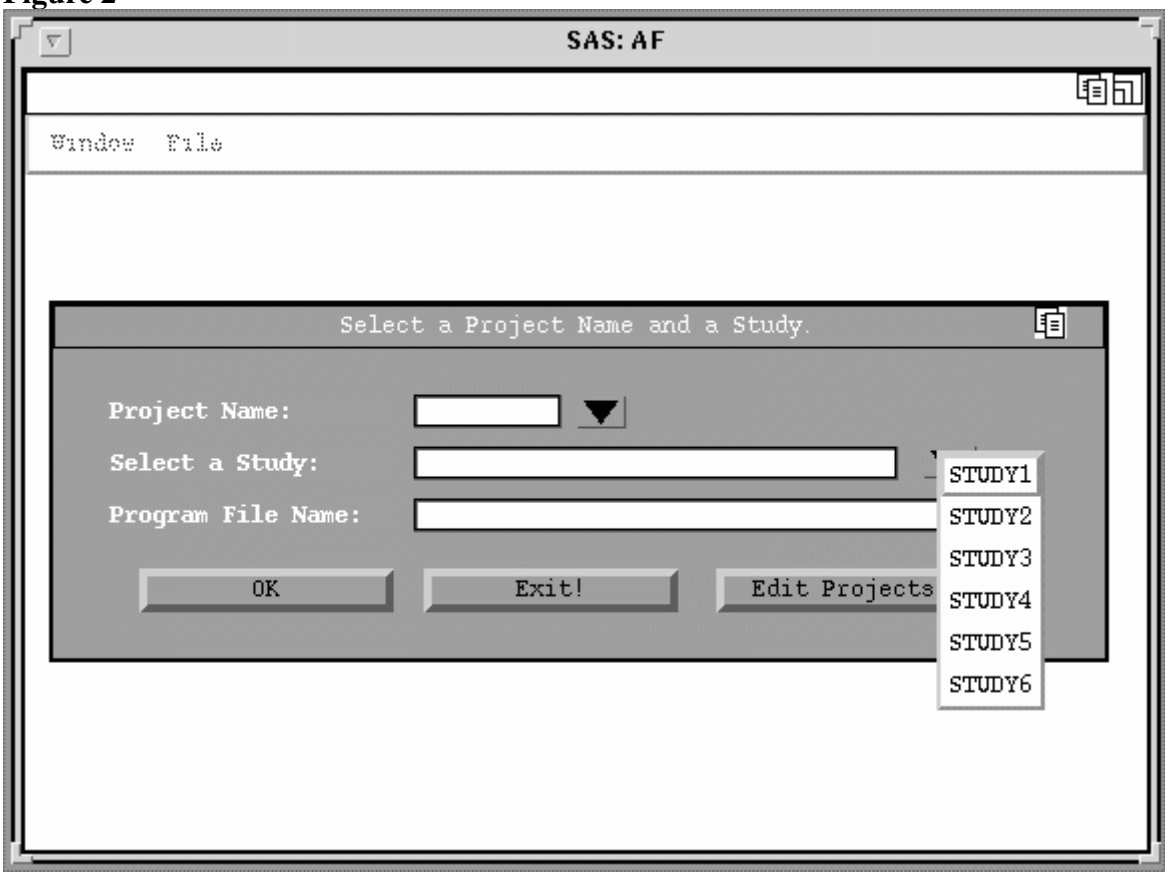

Figure 3

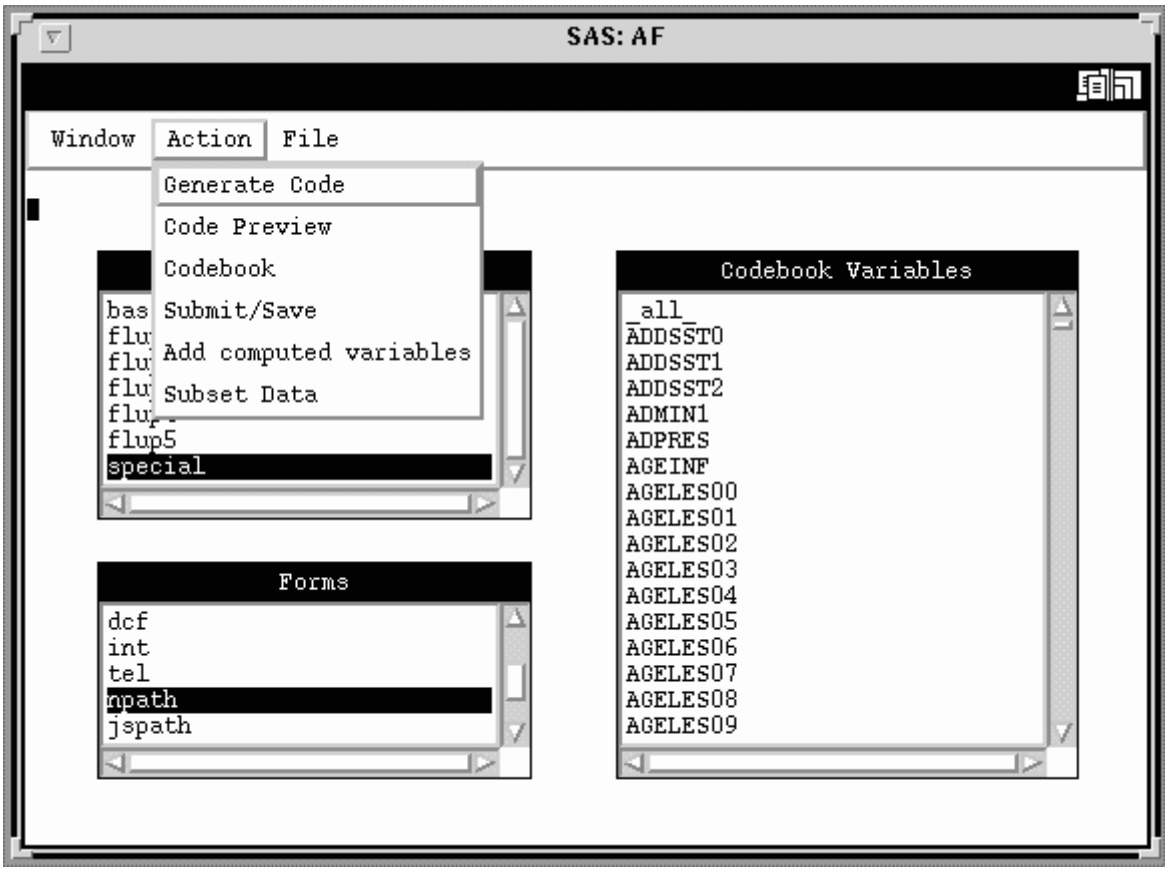

Figure 4

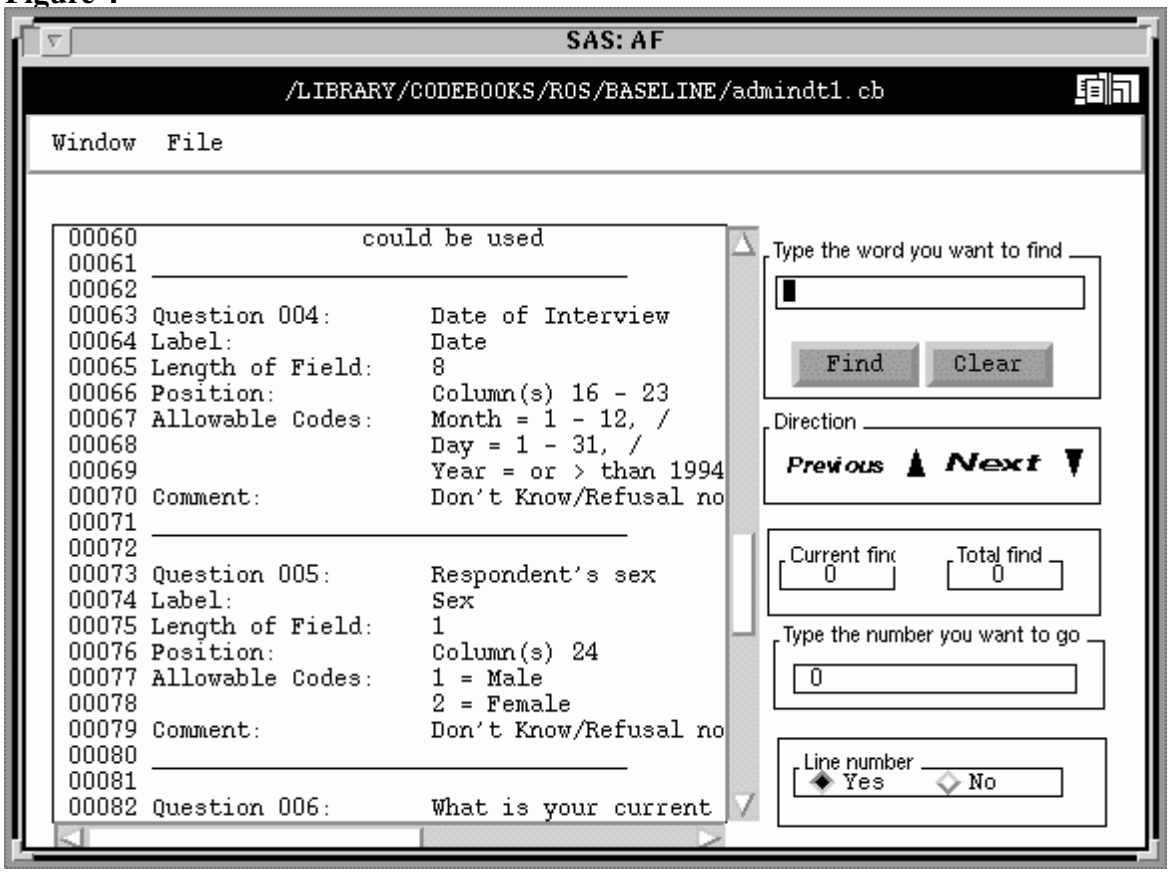

Figure 5

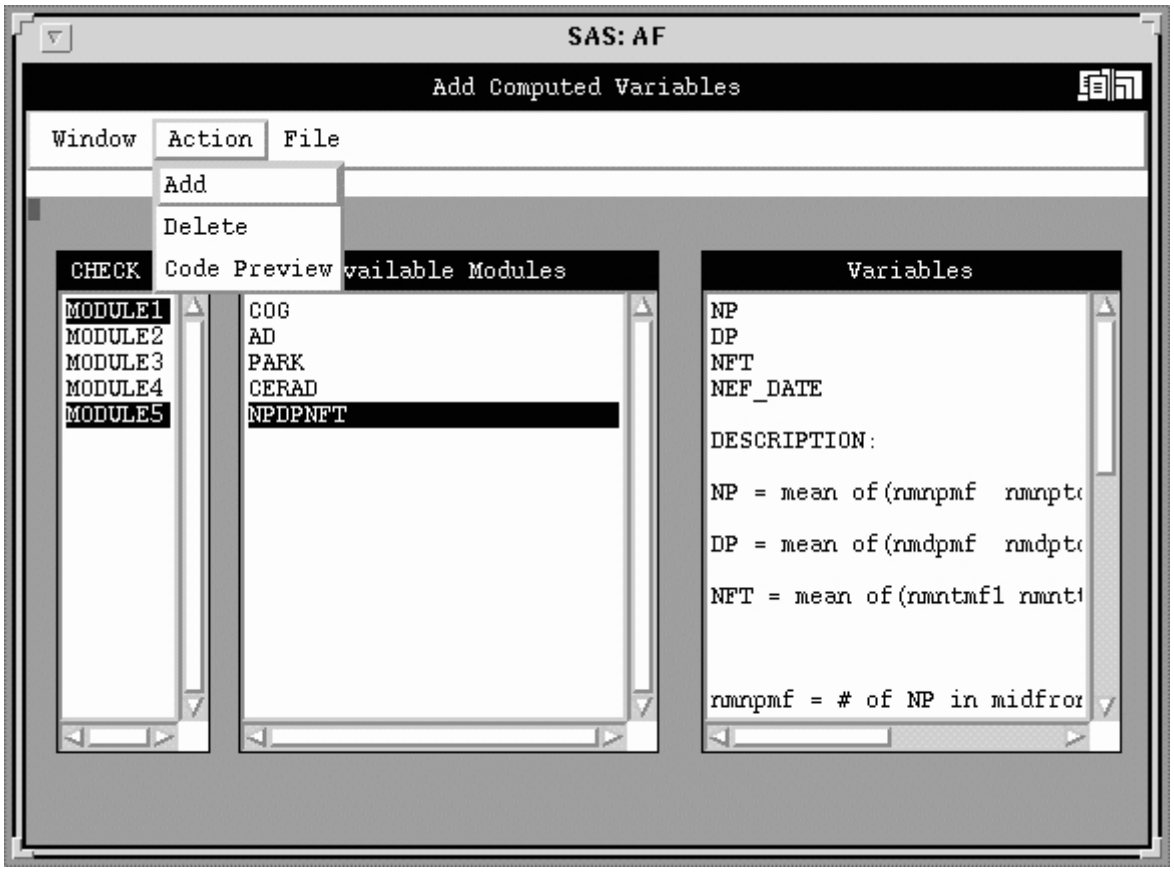

Figure 6

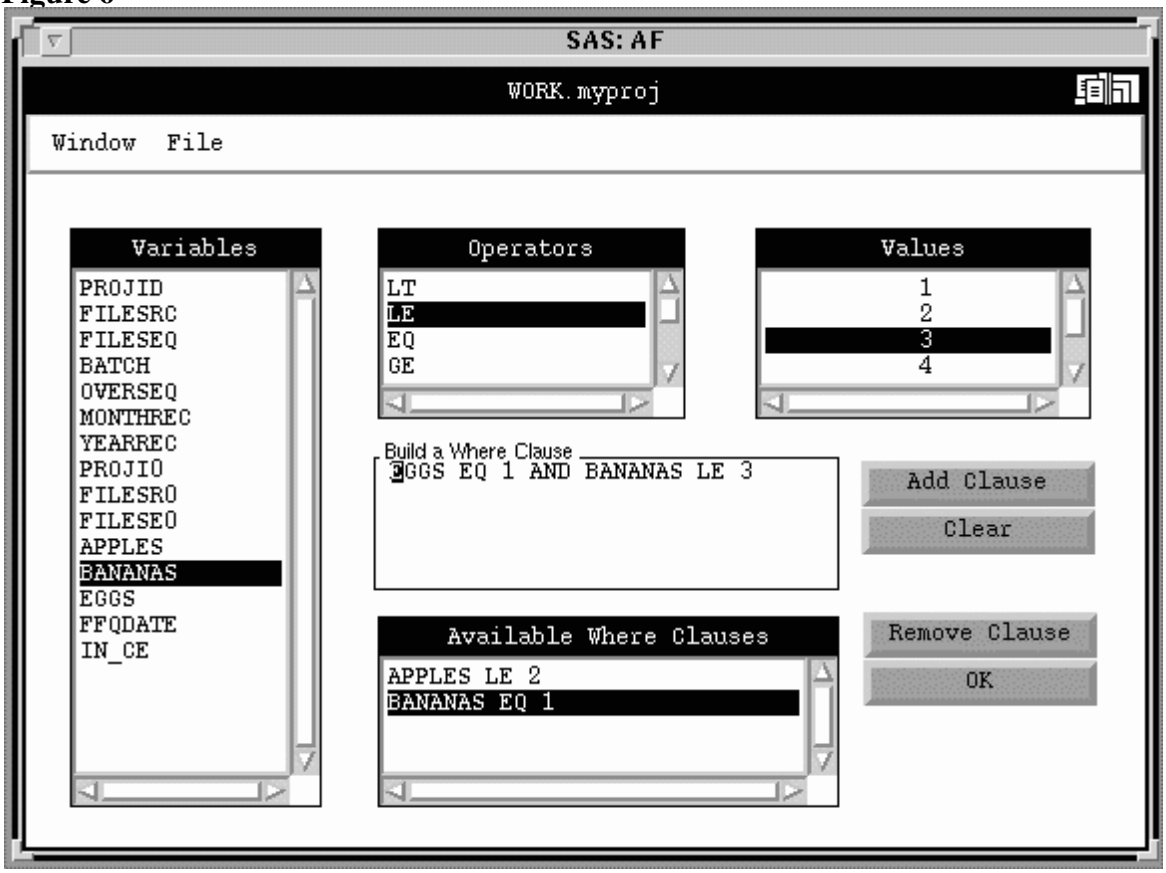

Figure 7