

Using SAS/AF® and SCL Extended Tables to Build a Custom-Designed Data Editor

Darius S. Baer, SAS Institute Inc.

Abstract

Although widely acknowledged as an industry standard for managing data and information, the **SAS® System** has come into its own as a sophisticated, easy-to-use application generator. Custom-designed data editors are in great demand. With the **SAS Application Facility (AF)** and **Screen Control Language (SCL)**, the developer can quickly prototype enduser requirements. As the enduser previews the application and recommends changes and enhancements, the developer can rapidly add and implement the additional code. The SAS System provides an environment in which the application requirements do not have to be totally formalized before development is begun. In fact, all that is needed is a general understanding of the business needs and the environment in which the application will be used.

General editors differ greatly from **Application Specific Editors (ASEs)**. In both, the user should be able to move freely around the screen scrolling up and down and left and right, entering data as is necessary. In a general text editor, however, the user is usually provided with only the tools needed to enter text and manipulate the text once it is entered. An Application Specific Editor, on the other hand, constrains the user into a formally defined environment which is easy to learn and fast to use. In addition, a well-designed Application Specific Editor will consist of a number of action buttons, pull-down menus, and pop-up selection screens. The goal is to develop a dynamic, syntax-free editor that verifies entries in real time and provides ultimate assistance in entering and updating data.

The above description of an Application Specific Editor was produced for a major U.S. railroad company in a train scheduling application. This paper will describe the tools that were used to build the editor. Specific techniques such as defining the desired extended table and associating the specified data set with the extended table will be covered. In addition, SAS/SCL techniques to manage data, verify fields, and pass data back and forth from the screen to the data set will be described. Once the user understands the four areas in which data exist in the application and how to move the data, the rest

of building the editor becomes easy. The four areas are the SAS Data Set, the Data Set Data Vector, the SCL Data Vector, and the Application Window. Although there are thousands of lines of code in this application, the major points can be reduced to a few fundamental processes.

Using SAS/AF and SAS/SCL to build an application-specific editor requires understanding the endusers' information needs and the data to be used. The basic framework for the editor is pre-defined. The developer formalizes the framework to the application requirements. Using the SAS System to build an editor is truly information delivery and decision support at its finest, both for rapid prototyping and for full application development.

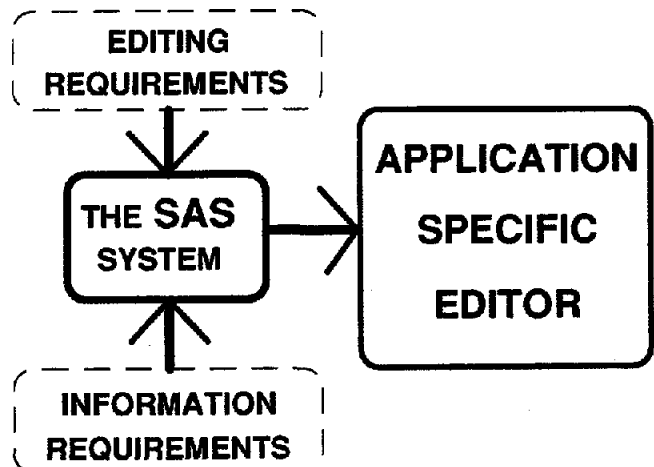


Figure 1. Information Delivery Platform

Introduction

The need for managing, processing, and analyzing data is never ending. Within an organization, individuals from data entry clerks to the CEO have requirements to access and deal with information. Automation and specialized presentation techniques can facilitate and enhance the efficiency of accessing and handling information.

An **Application Specific Editor (ASE)** is an environment that provides an efficient platform for editing and processing data and accessing information needed for decision making purposes (See Figure 1). This end-to-end information delivery

vehicle is now available for unique custom development totally within the SAS System.

Computer software and applications should be used to improve the **speed, usability, and validity** of data and information. What should be left to human beings are those activities which either cannot be done or can only be done inefficiently by the software. These include activities which require **intuition, analytic decision making** (that cannot be satisfactorily programmed) and even **data entry** when the data to be entered is not in a machine amenable format. As we learned early on, the computer and its applications are best suited for **repetitive activities** which not only reduce the drudgery for humans, but also reduce the potential errors. Additionally, the computer can process and present to the user in seconds or minutes **answers to problems** that could take days or weeks to solve by hand.

Applications should be **dynamic** or flexible enough to process the breadth of user requirements and **syntactic-free** or not requiring from the user correct typing of exact values. The user should be able to point-and-click their way through an application with minimum keystroke entry. By providing this type of user environment, the application designer and coder are truly meeting the needs of the enduser.

It was with these concepts in mind that an **Application Specific Editor (ASE)** was developed for a major railroad company to assist with their train scheduling activities. The application for the railroad was developed using Version 6.07 of the SAS System, and therefore uses character-based interactive screens rather than graphic-based frame technology available in later versions of the SAS System.

What is an Application Specific Editor?

In order to be considered to be an Application Specific Editor (ASE), an application must meet three criteria. First it must allow editing of data. Second, it must maintain those data in a row and column type format in a data set or database. Third, it must incorporate specific elements such as buttons, etc. that initiate actions required by the application. These actions can include counting, summing, searching, summarizing, presenting in a different format and problem solving. To some degree, an ASE can be thought of as an application with an editor rather than an editor that performs application actions. Whichever, it is the marriage between an application that provides solutions to problems and an editor that allows editing of the data needed by that application.

A common example of an ASE is a bookkeeping program that enables the user to access various accounts and presents the user with line items of the account to update, delete or add to. A store inventory system and a doctor's office system of patients are other examples of simple, generic ASEs.

In its most rigorous definition, an ASE allows editing of data in a row fashion with columns representing the variables while buttons, pull-downs and other icons add to the control of the data. For an analyst who may be trying to solve problems with the data in addition to editing the data, the application must incorporate analytic ability as well. Analysis often involves pattern recognition. The application can perform all necessary calculations and present information in a form that is best suited for interpretation by the analyst.

Application Specific Editor Characteristics

An Application Specific Editor (ASE) gets its name because it allows the enduser to edit data that is contained within a data set in application specific ways. The ASE enables constraints and prevents the user from making mistakes by enforcing proper syntax. Some enhancements include:

1. Action Buttons
2. Pull-down menus
3. Pop-up Selection Screens
4. Syntax Checking.
5. Semi-automated editing of data.
6. Data processing and manipulation.
7. Problem solving.

All ASEs should enhance the general approach that all applications undertake, that is to provide usability, speed, and validity checking. In addition, they provide a link or interface to a data set or database that is specific and essential to the application. It is with the data that is contained within the data set/database that answers to business/scientific problems are derived. These business/scientific problems are the foundation of constructing what we call drivers, that is, the specific items that must be addressed within the application in order to obtain answers or solutions to problems. The application, itself, takes on worth based on the degree to which it can satisfactorily address the drivers.

To build a successful ASE that will have value, it is critical to understand the requirements of the tasks that a user of the ASE has responsibility for. As an example, the user may be performing data entry. In that case, the application should reflect a pleasant, easy-to-use environment that runs reasonably fast,

and, as much as possible, validates the data as early as possible in the data entry process. There may be other activities that go on behind the scenes such as sorting data in chronological order, and action keys or buttons that allow for quick inserting and deleting of data. Information display in graphical and/or tabular format is also desirable. In the ASE environment, there should be a strong emphasis on solving user problems.

Satisfying the Application Requirements by Choosing the Right Tools

Because the SAS System has well-defined application development tools, it is rather straightforward for the developer to choose tools that are appropriate for the application. When building the ASE, it is useful first to study the application requirements and code the data steps and procedures that will produce the output the endusers are looking for. As the example in this paper, the building of an ASE for a major railroad will be used. In the case of the railroad, they were interested in having three objectives met.

First, the client wanted a graphic computer rendition of a system of trains with the X-axis representing time and the Y-axis representing distance. This graphic was called a Stringline and was used to ensure that trains did not cross or pass each other except at sidings and stations. Each individual line stands for an individual train schedule. Train schedules were already available in a computer file. SAS/GRAPH using PROC GLOT provided a perfect environment for displaying the graphic

requirements by enabling us to connect points with lines in a graph, we specified mileage points on the Y-axis and time on the X-axis so that different trains were represented by different colors within the same plot. See Figure 2. There was some processing of data that was necessary, but it was accomplished using straight-forward repeatable techniques. The Plot statement was simply Distance*Time=Train. Since the Time axis stretched from midnight for 24 hours, it was necessary to manipulate the data so that different 24 periods or days for a train were designated as subtrain numbers. The endusers were very satisfied with the ability to quickly view needed graphs.

The second desired item was the ability to derive the route and schedule that a train would take given the starting station, ending station, and a link file containing the distance and train speed allowed between adjacent stations. See Figure 3 for an example of a link entry which contains data for times and distances between nearest neighbor train stations.

LINKNUM	MILES	ORIG	DEST	TIMESP
92	54.35	01144	01089	0:52:00
93	49.33	01192	01144	0:45:00
94	75.00	01259	01192	0:58:00
95	87.33	01345	01259	1:15:00
859	54.35	01089	01144	0:47:00
860	49.33	01144	01192	0:45:00
861	75.00	01192	01259	0:59:00
862	87.33	01259	01345	1:20:00

Figure 3. Railroad Link File

Using Proc Netflow in SAS/OR and preparing the

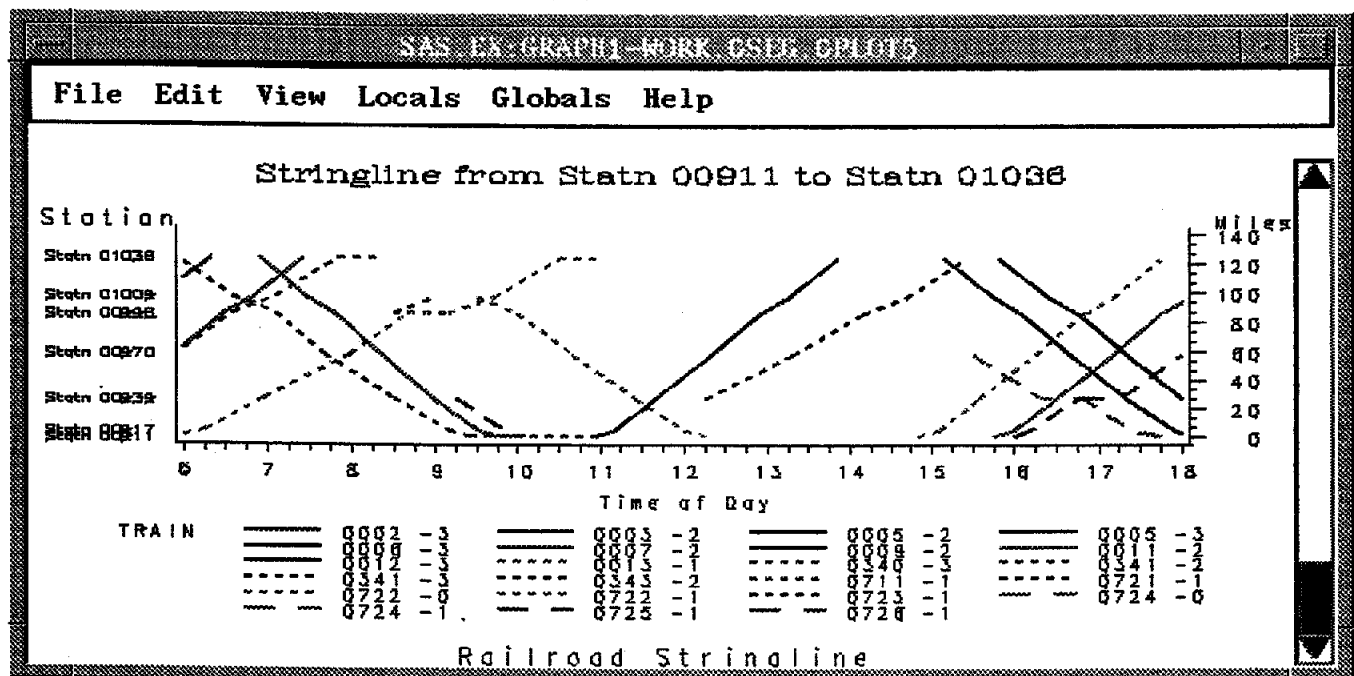


Figure 2. Stringline Graphic of Multiple Train Schedules within Time and Distance Constraints

data with Data steps, we were able to calculate the shortest distance or path between two points and report the stations that the train would travel through. From there, using a data step with constraints of departure and arrival times, a timed schedule could be produced as seen in Figure 4. Notice that the user can specify an arrival or departure time that will cause the train to be scheduled to go faster than its technically allowed speed. The Max Link Speed contains data that could be used in the application to prevent the user from scheduling a train to go faster than the Max Link Speed. In addition, using macro code, it was possible to iteratively run the NETFLOW procedure for a train that had required stops and would, therefore, link a series of shortest paths. Given the user-based and link file constraints, the code calculated what the arrival and departure times would be for all non-constrained stations along the route. As a matter of fact, given only an origin, a destination, a departure time, and the link file, a fully routed schedule could be and often was produced.

However, the railroad wanted the system to be interactive in that an analyst could make real-time changes to the schedule, have the system incorporate those changes, and then present the updated schedule back to the user. This interface would be powerful and easy-to-use for individuals not used to working with computers. Knowing that SAS/AF facilitated enduser interactive sessions, we began trying to implement the interactive application. In order to control the screen and the data being read from and written to the screen, we used the SAS Screen Control Language (SCL). The schedule for each train was stored in a data set where each row was a station in the route and contained arrival and departure times as well as the

distance that had been traveled from the preceding station. Of course, each train could have different numbers of stations in its route. With these requirements, we determined that the Extended Tables function of SCL was needed.

SAS/AF with SCL and Extended Tables enabled us to meet the third major requirement of the enduser while incorporating what we had done to fulfill the first two requirements. Using pop-up selection screens, the user wanting to display a Stringline can choose a starting station, an ending station, a starting time, and an ending time. The user can then visually analyze all the trains that travel through that space-time continuum. In order to tabularly display a schedule, the user only has to choose from a pop-up selection screen which train schedule they would like to view or edit. Lastly, there are a number of push-button functions that facilitate the editing and/or updating of a train's schedule. It is this ability which will be covered in more detail --- the capability to interactively edit a SAS data set within the confines of an Application Specific Editor (ASE).

Why Use the SAS System?

Because the SAS System is a fourth generation language in which you tell the system **what** you want done, not **how** to do it, we were able to build an ASE as a prototype in a fraction of the time that would have been required using a lower level language. All the tools necessary to satisfy the business requirements were totally contained within the SAS System environment.

The SAS tools that were used to build the Train Scheduling Application are as follows:

- Base SAS

Train Schedule Print for Train 0002 - TRAIN 0002 -- DAILY

STATION	Arr-ival	Depar-ture	D-ay	Dwell-Time	F F		Forc-Time	Sched-Time	Sched-Speed	Min Max		Link-Acum Mile	Link-Mile
					S	T				Link-Time	Link-Sped		
Statn 16001		20:30	1	0:00	Y	Y						0	
Statn 02203	21:07	21:07	1	0:00	Y			0:37	7.3	0:30	9.0	5	5
Statn 02200	21:13	21:13	1	0:00	Y			0:06	31.9	0:05	38.3	8	3
Statn 02195	21:25	21:25	1	0:00	Y			0:12	15.1	0:10	18.1	11	3
Statn 02182	21:44	21:44	1	0:00	Y			0:19	40.1	0:15	50.8	23	13
Statn 02172	21:56	21:56	1	0:00	Y			0:12	54.1	0:10	64.9	34	11
Statn 02169	22:08	22:08	1	0:00	Y			0:12	21.8	0:10	26.2	39	4
Statn 02166	22:22	22:22	1	0:00	Y	Y		0:14	3.5	0:10	4.9	39	1
Statn 02165	22:31	22:31	1	0:00	Y			0:09	8.9	0:10	8.0	41	1
Statn 02158	22:45	22:45	1	0:00	Y			0:14	30.6	0:15	28.6	48	7
Statn 02044	3:14	3:14	2	0:00	Y			4:29	27.8	4:45	26.2	173	125
Statn 02038	3:33	3:33	2	0:00	Y	Y		0:19	22.4	0:20	21.2	180	7
Statn 02009	4:15	4:15	2	0:00	Y	Y		0:42	41.5	0:45	38.8	209	29

Figure 4. Selected Train Timed Schedule Printed for User

- Used to access and manage the data.
- **SAS/GRAPH — PROC GPLOT**
Used to graphically display the train schedules that traveled the same route within the same time period.
- **SAS/OR — PROC NETFLOW**
Used to calculate the shortest path for a train route between any two stations in the system.
- **SAS/AF**
Provides an interactive environment in which the end user can select specific data to process and specific sub-applications to run.
- **Screen Control Language**
Used to control the screen environment, establish the editing capabilities, and link the screen variables to the data set variables -- controls the attributes, placement, and relationship of the variables used within the interactive SAS/AF environment.
- **Extended Tables**
Used as the integral part of the system that controls and displays the data set that the enduser edits and views. Provides an interactive, customizable, dynamic window into the data of interest.
- **SAS Macros**
Used to iteratively run the processes created by the other tools.

Together, SAS/AF, SAS/SCL, and Extended Tables provide a totally push-button, interactive environment, customizable to the specific needs of the enduser that can be used for a wide variety of data analysis, what-if, and decision support requirements. With appropriate design and coding, the enduser can request that the edited data set(s) be processed and redisplayed in both tabular and graphical form.

Because of the ability to perform rapid prototyping, we were able to make changes to the application which we thought of during the prototyping, itself. These changes were easily added to the application without too much effort. Some companies consider using the SAS System to build a prototype and other lower level languages to implement the production code. However, the SAS environment used for prototyping is so structured and reliable that the endusers often choose to develop the production ASE using the SAS tools as well.

Since the SAS tools already existed, we only had to tailor them to the needs of the client. The remainder of the paper will describe how we accomplished putting together the ASE portion of the application realizing that the application consists of a number of menus, selection screens, and the STRINGLINE procedure. The result was an application specifically implemented

as requested and designed in minimal time and effort because of the use of appropriate approach and software tools.

Application Implementation Details of the Train Scheduling Example

What makes the train scheduling application an editor is the capability provided to the user to modify various fields within the rows of the train schedule. Figure 5 shows the Train Scheduling ASE from the user's perspective. The way in which these editor functions are implemented using the SAS processes is a major focus of this paper.

As this paper's primary emphasis is on SAS Application Facility, Screen Control Language, and extended tables, the implementation details will be limited to that aspect of the application. When we began to create the interactive capability for the end users, we used PROC BUILD within SAS/AF. We created a small menu system that upon user request would display the graphic Stringline between 2 chosen stations or produce a tabular train schedule print in the output window for a chosen train.

From that point, we began the development to put together the total ASE. One of the best parts of the process was being able to build the requirements into the system as the client came up with them. The ability to have a feedback loop of this nature in the development process is unusual in most application development environments. With the SAS System, general applications can be created and shown to the client which will often spark new ideas about requirements which can then be implemented, etc. Figure 6 illustrates the application development process as described.

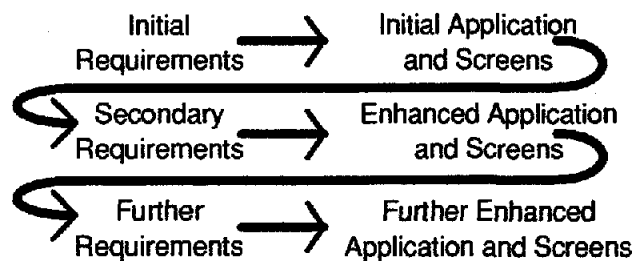


Figure 6. Feedback from Client to Developer

Using SAS/SCL

Each program using SAS/SCL has 3 major sections: the INIT, MAIN, and TERM sections. These sections initialize, repetitively process and terminate the SCL program. There are also two small, but very essential sections for managing the extended table. These are the GETROW and PUTROW sections.

SAS AF
Train Schedule Editor

TRAIN NUMBER: 0002

TRAIN 0002

DAILY

RETURN

UPDATE

ADD LINE AFT

COPY TRAIN

PRINT

FORCE

SAVE

DEL CURLINE

DELET TRAIN

STRINGLINE

TIMEZONE

Station	Arri- val	Depar- ture	T Z	D- ay	Dwell Time	D S	S F	F F	K T	e T	Force Time	Sched Time	Schd Spd	Min L	Max Link	Link Spd	Acum Mile	Link Mile
STATN 00001		20:30		1										2				
STATN 02203	20:40	20:40	P	1	0:00				Y	Y	Y	0:10	26.3	2	0:10	27.0	5	5
STATN 02200	20:45	20:45	P	1	0:00				Y	Y	Y	0:05	37.3	2	0:05	38.3	8	3
STATN 02195	20:55	20:55	P	1	0:00				Y	Y	Y	0:10	17.7	2	0:10	18.1	11	3
STATN 02182	21:11	21:11	P	1	0:00				Y	Y	Y	0:15	49.5	2	0:15	50.8	23	13
STATN 02172	21:21	21:21	P	1	0:00				Y	Y	Y	0:10	63.2	2	0:10	64.9	34	11
STATN 02169	21:31	21:31	P	1	0:00				Y	Y	Y	0:10	25.5	2	0:10	26.2	39	4
STATN 02166	21:41	21:41	P	1	0:00				Y	Y	Y	0:10	4.7	2	0:10	4.9	39	1
STATN 02165	21:52	21:52	P	1	0:00				Y	Y	Y	0:10	7.8	2	0:10	8.0	41	1
STATN 02158	22:07	22:07	P	1	0:00				Y	Y	Y	0:15	27.8	2	0:15	28.6	48	7

Figure 5. Train Scheduling Screen from the User's Perspective

A very powerful paired command used in SCL is the SUBMIT CONTINUE and ENDSUBMIT statements which facilitate processing of regular non-SCL SAS code. It is within the confines of the SUBMIT block that we preprocessed the data and fulfilled the user requests for rescheduling and rerouting the train. We would then redisplay the processed data set for the user's examination. In order to present the train schedule data set to the user for viewing, scrolling, and editing, we used the extended tables feature of SCL which will be discussed next.

Extended Tables

A major element of editing a data set within the SCL environment is using Extended Tables. Within an extended table, the user can scroll up and down, modifying data as is allowed by the attributes of the variables. A major feature is that an initially unknown number of observations can be handled in the extended table. We used three basic steps to implement the extended table in our application. In the screen, we specified where the identifier and data set elements would be stored. Figure 7 illustrates the Train Scheduling Screen from the developer's perspective. By using three carets (^) in columns 1-3, we were able to begin the extended table in the row immediately following the ^^^. In the attribute table, we named the alias of each field

by the same variable name that it was known by in the data set. This allowed the screen to be updated when the data set changed and the data set to be updated when the screen changed. The power of this previous statement cannot be overstated.

In order to control the extended table, the GETROW and PUTROW sections are required and are placed immediately following the TERM section. The GETROW section contains statements that are executed repeatedly until the display window is full of observations. In our application, the GETROW section contains the single statement:

```
IF (FETCHOBS (RCREATID, _CURROW_) = -1)
  THEN CALL ENDTABLE();
```

where FETCHOBS reads a specified observation from a SAS dataset into the Data Set Data Vector and by doing so, makes the observation available to the SAS/AF application.

The PUTROW section is used to cause the dataset to be updated when the fields in the extended table on the screen are changed. Our PUTROW section contained two statements: 1) The FETCHOBS function with the NOSET option so that the SCL variables were NOT updated with the values from the SAS data set because this activity is done in the MAIN section as described later. 2) The UPDATE function which writes

```

&MSG_____
&TTN_____&TNAME_____&FREQRN_____&RETUR_____
&UPDATE_____&ADD_____&COPYTR_____&PRINT_____&FORRC_____
&PERM_____&DEL_____&DELTR_____&STRING_____&TIMEZZ_____

                D S F F K                C Min    Max
Station  Arri- Depar T D- Dwell W T S T e Force Sched Schd L Link  Link Acum Link
^^^^      val  ture Z ay Time F P F F y Time  Time  Sped S Time  Sped Mile Mile
&STATION__ &AR__ &DE__ & &D &DWEL & & & & &FORC &NEWT &SP_ & &TIM_ &MAX &TML &MIL

```

Figure 7. Train Scheduling Screen from the Developer's Perspective

values from the Data Set Data Vector back into the SAS data set.

After anything on the screen is modified, the PUTROW, MAIN, and GETROW sections are executed in this order. Although this process may seem very complicated, a lot of data management from screen to data set and back is being handled by the SAS System and the application programmer need know only which statements to use and where to place them in the code.

The Attribute Section

The attribute section specifies the link between the screen variables and the data set variables. It also specifies how the screen variable will be used and what their characteristics are. For example, see Figure 8 to see the attribute screen for the train station and arrival time variables. We specified the alias which most often should be a link to a SAS data set variable. The major characteristics used in the attribute section can best be learned by reading the SAS Application Facility text and/or taking a

SAS/SCL course. In the general attributes, we specify that we are using an extended table.

SAS/SCL Functions

Lastly, we opened the data set for use by the following 3 statements:

- RCREATID = OPEN('CALCTEMP','U');
- CALL SET(RCREATID)
- CALL SETROW(0,1,'','Y');

These three statements are so important that they will be explained in detail. The first statement uses an SCL variable, in this case RCREATID as the SCL variable to be referenced in order to access a SAS data set interactively. This statement simply opens a SAS data set for update and provides a data set identifier. The CALL SET statement links the SAS data set variables with SCL variables of the same name and type. Specifically, CALL SET creates a link from the screen variables to the SCL data vector, from there to the SAS data set data vector, and finally, to the SAS data set, itself. In this way, the user

```

Field name: STATION  Frame: 2  Row: 1  Col: 1  Length:
Alias: STATION      Choice group:
Type: CHAR          Protect: YES
Format:             Just: LEFT
Informat:
Error color: RED    attr: REVERSE  Help:
List:
Initial:
Replace:
Options: CAPS AUTOSKIP
-----
Field name: AR       Frame: 2  Row: 1  Col: 13  Length:
Alias: ARR          Choice group:
Type: NUM           Protect: NO
Format: TIME5.      Just: RIGHT
Informat: TIME5.
Error color: WHITE  attr: REVERSE  Help:
List:
Initial:
Replace:
Options: CAPS AUTOSKIP

```

Figure 8. Attribute Screen for the Station and Arrival Time Variables

editing the screen data is actually making changes to a SAS data set on disk. The CALL SETROW statement defines an extended table, in our case, a dynamic one in order to use the extended tables feature for an initially unknown number of observations.

Because of the nature of the environment, prior to every processing of the data within a SUBMIT block, it was necessary to close the link between the screen variables and the SAS data set, and then reopen the link after the processing was done. This was done by using the statement:

- `RC = CLOSE(RCREATID);`

Once the INIT section has been processed, the user is free to edit the displayed data and execute any available commands. If any data management or modification is necessary, the screen will be disconnected from the data set, the data processing will occur, and then the data set and screen variables will be reconnected. All of this goes on behind the scenes without the enduser needing to understand the mechanics.

Every time we process the data for a train, we close the link between the screen and the disk data set. Then we process the data, relink the variables, and present the data in its updated form to the user.

User Application Actions

In the data portion of the screen, the user could perform many editing functions. They could add or delete a station in the route, modify the arrival and/or departure times, specify an amount of time to dwell at a station, and force a train to take more or less time traveling from the previous station. Also, the user could modify certain flags that would be used in processing the data, such as if a train must stop at a certain station along the route and if the specified arrival and departure times were fixed by the user and could not be adjusted by the scheduling code.

A couple of key points should be mentioned here. First, certain fields were specified as non-editable. The application calculates the values and they exist for viewing, only. Second, those fields that are modifiable could be changed within constraints. Lastly, as an

example, the station field could NOT be modified even though stations could be added or deleted through the use of a pull-down menu or designating the row of the route, respectively. Any row could be selected for editing or processing by simply placing the cursor on the row and clicking the left mouse button.

In the MAIN section, the program waits for the user to choose an action. In this case, there are 11 possible actions. They will be covered in the order that they appear on the screen in figure 5. Each of the 11 actions can be initiated by pressing the left mouse key when the mouse cursor is on the desired action button on the screen. All of the actions were set up to belong to what is called in SCL a Choice Group. The choice group for all 11 actions is called BUTTON1. Using an IF statement within the SCL code, the program can ascertain whether a button has been selected. When a button is selected such as RETURN, the choice group BUTTON1 takes on the value listed in the LIST section of the attribute for the SCL variable RETURN which in this case was given the value 'YES'. See figure 9 for a display of the attribute information for the variable RETURN.

When RETURN is chosen, the link to the schedule data set is closed and the user is returned to the program that called the train schedule editor by setting the SCL system variable _STATUS_ to 'H' as shown in the following SAS statements:

```
IF BUTTON1='YES' THEN DO;
  IF (RCID>0) THEN RCD1=CLOSE(RCID);
  PUT RCD1=;   _STATUS_='H';   RETURN;
END;
```

In the returned-to program which will not be shown, the user is able to select which train schedule they want to view and/or edit. For all remaining 10 actions, BUTTON1 is reset to blank in the executed DO loop for those actions.

When the UPDATE key is pressed, the choice group BUTTON1 takes on the value UPDATE and the section of code within the IF statement, IF BUTTON1 = 'UPDATE', is executed. This process takes as much as 1 minute so a processing message is displayed to the user. Any remaining changes to the screen are updated in the data set with the SCL UPDATE function, the link from

Field name:	RETUR	Frame: 1	Row: 3	Col: 63	Length:
Alias:	RETUR	Choice group:	BUTTON1	Pad:	
Type:	CHAR	Protect:	INITIAL		
Format:		Just:	CENTER		
Informat:					
Error color:	RED	attr:	REVERSE	Help:	
List:	YES				
Initial:	RETURN				
Replace:					
Options:	CAPS AUTOSKIP				

Figure 9. Attribute Screen for the Return Variable

the screen to the data set is closed, and the data set processing occurs. This processing includes complex data set manipulations to reroute and/or reschedule the train. Multiple invocations of PROC NETFLOW within a macro are used to find the shortest paths between required stations along the route. This logic requires only 400 lines of SAS code within which an individual train is totally routed and scheduled, constrained by system rules of the railroad and user constraints set by user interaction through the editor. The system chooses the stations and sets appropriate arrival and departure times based on the agreed to algorithms. When the UPDATE function completes, the link from the data set to the screen variables is reestablished and the enduser can continue to modify or save the results permanently.

When the user presses the ADD LINE AFT key and as long as the user has selected a row following which the new station is to be added, a pop-up selection screen of stations is displayed through the use of the DATALISTC function shown in Figure 10. The statements used are:

- RNODE=OPEN('TRAIN.NODE');
- SELNO=DATALISTC(RNODE,'DEST STATION TZ','SELECT ONE _NODE_');

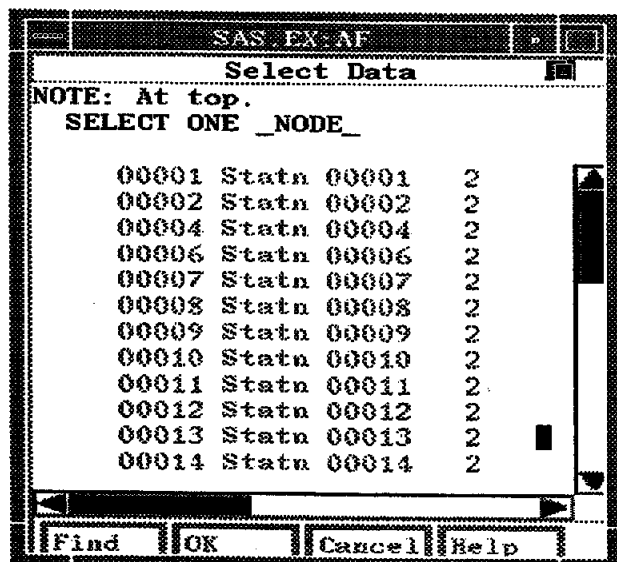


Figure 10. DATALISTC - Station Selection

SAS/SCL provides this capability by creating a link between what is shown in the pop-up screen and an auxiliary SAS data set. The listed stations are actually cities and towns in the real application, but have been changed for this paper to reflect the destination number. Assuming the user selects a listed station, associated data is accessed for that station from the file, the screen link to the schedule data set is closed, the station is inserted into the train schedule data set, and the link between the screen and SAS data set is reopened.

If the user chooses to create a copy of a train schedule with a different name, the user selects COPY TRAIN and a subprogram is executed in which the user picks an unused train number from a list produced by the program, and specifies a name to attach to the train number. Once again, the screen — SAS data set link is closed, the SAS data set is updated, and the link is reestablished.

Printing is the same as updating except that the user is provided with a number of hard and softcopy output options through the use of the SHOWLIST function. When the user selects SAVE, the changes, which up to this point have been stored in a SAS WORK data set, are copied into the permanent train schedule data set. The FORCE key allows the user to force the reroute portion of code to be executed in the UPDATE subroutine. Normally, this code is only executed when the route has been modified with an add station or delete station function.

The DEL CURLINE function operates on the station that is currently selected and removes that station from the schedule. In rerouting the train, however, the removed station will be brought back in to the schedule if it is on the shortest path. The DELET TRAIN key allows the user to remove the data for the train being viewed from the permanent train system schedule data set. The STRINGLINE key makes a call to the stringline program where the user can graphically display those trains that travel within the space and time that is selected. Finally, the user can select TIMEZONE in which a pop-up screen as shown in Figure 11 is displayed through the use of the SHOWLIST function. The user chooses which of 4 time zones they want the schedule displayed in or local time. The SHOWLIST program statement is:

- WHTZ=SHOWLIST('Local Time', 'Pacific Time', 'Mountain Time', 'Central Time', 'Eastern Time', 'Choose a Time Zone');

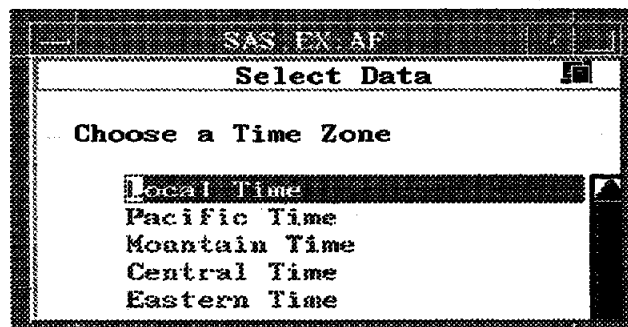


Figure 11. SHOWLIST - Timezone Selection

Again, the screen—data set link is closed, the data set is processed, and the link is reopened with the updated data displayed.

This detailed explanation of the functions was used to demonstrate the wealth of capability available within the SAS/AF, SAS/SCL, and Extended Tables environment. The following SAS/SCL commands and functions were used to achieve the results:

SAS/SCL Function Summary

- The OPEN, SET, & SETROW functions of SCL to create a link from the screen to the SAS data set.
- The GETROW and PUTROW sections to establish the fields in the extended table and to execute for each logical row modified by a user, respectively.
- The CLOSE function of SCL to close the link between the screen variables and the SAS data set. This was used whenever processing of the SAS data set needed to be done.
- The use of the ALIAS field in the attribute screen for a screen variable. As long as it matched the SAS data set variable name, the data associated with the data set variable would automatically be brought into the screen variable and displayed.
- The SUBMIT CONTINUE — ENDSUBMIT block which was used to execute SAS code such as a DATA step or Procedure.
- The DATALISTC function to display a list of observations such as a list of train stations from which one could be selected to be added to the schedule.
- The SHOWLIST function to display constant items from which the user could choose such as which time zone to display the schedule in or which options were available for printing the schedule.

Summary

The single important message to be gotten from this application is that SAS has provided a straightforward environment in which the enduser can access, edit, and apply specific action to data sets without needing to be a programmer. The SAS programmer is able to quickly and efficiently implement the user requests because powerful tools have been provided to manage the enduser environment.

This application demonstrated the use of SAS/AF using PROC BUILD, the SAS Screen Control Language, and Extended Table capabilities. The detailed review of the train scheduling application provides a template upon which other applications can be patterned. One of the most interesting phenomena observed while developing this application was noting the degree to which the enduser became aware of application capabilities that originally they had not thought were possible, but in fact, could be readily implemented in the SAS System with tools constructed

specifically for those purposes. Modification of the timezone was one example.

It is the need to manipulate data in application specific ways that provided the concept of building an Application Specific Editor. Understanding what is needed to produce an ASE means that an application designer can quickly sketch the outline of the application given a few requirements. From that point on, the application requester, designer, and coder work as a team to produce the prototype, and finally, the production application. This process is a smooth flowing one with maximum communication and minimum mistakes. The SAS System provides an environment with powerful tools, a great number of preprogrammed procedures resulting in fewer coding errors, minimal application programmer code required for functions provided, and the ability to quickly test and debug code.

Although the requested functions of this application could be coded in other lower level languages, it is the power and breadth of capability of the SAS System that enables the application developer to rapidly prototype programs to be used to produce information for decision making purposes.

As most business and scientific arenas have a need for an ASE, this paper serves as an excellent example for emulation when designing and building an ASE which can be accomplished totally within the environment of one language, the SAS System.

Bibliography

1. SAS Institute, Inc., SAS/AF® Software: Usage and Reference, Version 6, First Edition, Cary NC: SAS Institute Inc., 1989. 245 pp.
2. SAS Institute, Inc., SAS® Screen Control Language: Reference, Version 6, First Edition, Cary NC: SAS Institute Inc., 1990. 464 pp.

For more information, contact:

Darius S. Baer, Ph.D.
SAS Institute Inc.
1950 Stemmons Freeway
Suite 5052
Dallas, TX 75207-3109
(214) 746-3916

*SAS, SAS/AF, and SAS/SCL are registered trademarks or trademarks of SAS Institute Inc., Cary, NC USA