Abstract

The relational nature of a SAS database often puts it at an advantage over a hierarchical database for ease of access and maintenance. However, continually updating a SAS database can slow turnarounds, particularly as the size of the database increases. To reduce access time we developed a procedure which transfers data directly from a SAS database to an ISPF table. This paper assumes the availability of ISPF (Interactive System Productivity Facility) and the 'Dialog Management Services' required to develop an interactive application under ISPF.

Introduction

In applications which involve regularly updating a SAS database, minimizing the response time per transaction becomes a priority. This is particularly true when the update is processed interactively. This was achieved in the present application through the use of an interface between SAS and IBM's ISPF (Interactive System Productivity Facility). Proc SPFLINK provides the link necessary to transfer a SAS database to an ISPF table which can then be used under the supervision of the dialog manager. A SAS database can thus be retrieved and modified without requiring the user to interact with SAS, and with the speed and convenience which an ISPF table provides. A new SAS dataset is then produced from this table and is ready to be accessed. An overview of the overall system is provided below.

History

Because of its numerous advantages, a SAS database was used in the present application, even though it was necessary that the database be frequently updated. A first attempt to expedite the update transaction involved using a CLIST to create a transaction record from a user's desired changes, and to then call SAS to modify the database with the transaction using, for example, the update procedure. The drawbacks of this approach were: the slow turnaround time and the effort involved in verifying the transaction.

A more efficient solution in updating a masterfile involves the use of ISPF. Panels input data, verify it and temporarily store records in ISPF tables. This would increase the speed, and ease of data entry and validation. The table is like a two dimensional array, where columns correspond to variables and rows contain the respective values for those variables. They may be dynamically updated during execution. Tables provide a very useful tool for updating a master file, but we still wanted to use a SAS dataset for processing the data.

This is why in this application there was a need to first transfer information from a SAS dataset to an ISPF table, update it, and transfer it back again to a SAS dataset. Although we could achieve this by copying data into a temporary sequential file as an intermediate step, this would again result in an unacceptable turnaround time when frequent updates are necessary, and would not be efficient. To eliminate this step, a program invoked through SAS, Proc SPFLINK, was created to enable a direct link between SAS and ISPF.

Description

Proc SPFLINK was developed using PL/1, a language which is familiar to many SAS users. The ease of interaction with SAS and ISPF made this language best suited for this application.

The program uses SAS interface routines (as described in the SAS programmers guide (1981) edition and ISPF Dialog Services (as described in Dialog Management Services)).

This program consists of two main subroutines which constitute the two program options: (see Figure 1)
1. Transfer variables from a SAS Database to an ISPF table (SASTOSPF).

2. Transfer variables from an ISPF table to a SAS Database (SPFTOSAS).

Before the subroutines are invoked, two preliminary steps are necessary:

**Step 1:** Specify all the SAS interface routines needed for the procedure (see program listing). For example,

- **INPUT** (advances to next observation in SAS dataset)
- **VARY,VAR** (transfers character and numeric variables to procedure)

**Step 2:** The external subroutine, called ISPLINK, must be declared, to enable the program to invoke the ISPF Dialog Management Services required for the creation of the table, and to transfer all variables from the SAS dataset. Such services are: (summarized from ISPF Dialog Management Services reference manual)

- **VDEFINE:** invoked by the program to link or give ISPF the ability to directly access variables used within the program
- **TBCREATE:** create a new table in virtual storage and open it for processing
- **TBADD:** to add a new row of variables containing the current observation values to a table

All these services are fully explained in the appropriate ISPF manuals. It is suggested that a general knowledge of this IBM program product be acquired before attempting to develop this function.

At this point, the SASTOSPF or SPFTOSAS subroutine can be called.

**SASTOSPF**

As its name supposes, this subroutine creates an SPF table from a SAS dataset. The main steps of this subroutine are:

**Step 1:** All variables contained in the SAS dataset must be declared in the program. The character values are declared as such, numeric variables are defined with attribute FIXED BINARY(31), a full word. Numeric variables must be represented this way to be used within ISPF and ISPF TABLES. As mentioned before, the variables in the dataset must be read using SAS interface routines, and since most of the arguments of these routines are FLOAT BINARY(53) data format, it is necessary to declare this type of variable over the previously declared CHARACTER and FIXED BINARY variables. It is then this FLOAT BINARY variable which is passed as an argument to the routine. This is not true when referring to the variable at any other time. This is an important consideration to keep in mind throughout the program.

**Step 2:** A link must be made between the variables used within the program and the names by which they are referenced by ISPF to access the data. This is accomplished by using the ISPF Dialog Management Service **VDEFINE**. The format is:

- **CALL ISPLINK('VDEFINE',SPF_VAR,PGM_VAR,TYPE,LENGTH);**

where:

- SPF_VAR is the name of the variable by which SPF will address a particular variable. This variable name must be enclosed in parentheses.
- PGM_VAR is the name of the variable which corresponds to the SPF_VAR, this is the variable name used in the program
- TYPE refers to the variable type: 'CHAR' implies a character variable
- 'FIXED' refers to a numerical variable
- LENGTH: this parameter must be a full word (FIXED BIN(31)); it cannot be passed as a constant. The length for a character variable is the variable's actual length, while numeric variables may have a maximum length of 4 bytes.

**Step 3:** Create the ISPF table. This is done by invoking another SPF service, **TBCREATE**. The format is:

- **ISPLINK('TBCREATE',TABLE_NAME,KEY, VARIABLE_LIST,DISPOSITION)**

where:

- TABLE_NAME is the SPF table being created.

A variable in the ISPF table will serve as the KEY to access rows and retrieve them.

**VARIABLE LIST** consists of the variables, other than the key, which will make up a row in the SPF table.
were read from the SAS dataset and transferred to the newly created SAS dataset. This process was repeated until all observations contained in the existing SAS dataset.

The format is:

```
CALL ISPLINK ('TBADD', TABLE NAME);
```

Since only integer values can be stored in the table for numeric values, these variables, must first be converted to integer. The SAS dataset in the present application contained variables with two decimal places, and to store them in the table, they were multiplied by 100.

This process was repeated until all observations were read from the SAS dataset and transferred to the ISPF table.

Once the ISPF table is created, it is then possible to update it. In this application, CLISTS were used to display the required panels and to update accordingly the ISPF table. Information is then added, modified or deleted.

**SPF TO SAS**

When all transactions have been processed, the ISPF table must be converted back to a SAS dataset. To accomplish this, other SAS routines are used to create a new SAS dataset, to copy variable names and attributes from the existing SAS dataset, and output the ISPF table to the newly created SAS dataset.

**Step 1:** Define the new variables for the output dataset. This is accomplished by copying them from the existing SAS dataset. The routine NAMEV is used to read and store complete information of a variable contained in the existing SAS dataset. The information such as name, type, position of variable in observation, and length are stored in the structure NAMESTR. Using the routine DNAMEV it is then copied into the output SAS dataset being created. Before variable information can be copied, the routine LOADOUT must be called, which makes available the output routines needed to copy the information. Since two SAS datasets are now involved in the procedure, the SETDSN routine is used to specify which dataset is being processed at the current time.

**Step 2:** The output dataset is opened for processing. A vector called OUTVEC is used to store instructions for moving the data. The first element of this vector must be initialized to zero, since this is assumed by the SAS routine using it. When an array is used, its size is determined by the number of variables to be written to the output dataset. The rule is:

\[
\text{# of elem.} = \text{CEIL}\left(\frac{42 + \text{# of vars.} \times 6}{4}\right).
\]

**Step 3:** When all variable information has been copied, each row in the ISPF table is transferred to the SAS dataset. Since the observations are created in the program and not read from the input dataset, a structure is used to hold all the variable values. All the character elements of this structure correspond with the variable names in the ISPF table, while the numeric values must be declared with attribute FLOAT BINARY, since this is how numeric data is stored in a SAS dataset. These elements also have the attribute UNALIGNED to prevent PL/1 from aligning them on storage boundaries. This is important because the location of each element corresponds to the location of the variables in the SAS dataset. This is also why there is an extra element at the beginning of the structure. It's used to offset the position of the next element to location 5, the same position at which the first variable is located in the SAS dataset.

Each row is then read, and each value in the SPF table is transferred to its corresponding variable in the program. The character values are directly stored in the elements of the structure to be written to the SAS dataset, while the numeric values, stored in FIXED BIN variables, must be transferred to FLOAT BIN elements of the structure. If necessary, they must be divided by 100 to get the decimal values (recall that SPF table could not store decimal places and therefore values were multiplied by 100 to get integer values).

**Step 4:** The missing values contained originally in the SAS dataset were transferred to the SPF table as zeros and must therefore be changed back to missing when writing the observation to the SAS dataset. In this example, this was accomplished by declaring a bit string variable of length 8 on the FLOAT BINARY variable. If the variable has a zero value when it should be missing, then the bit string variable (and at the same time the FLOAT BINARY variable) is set to '10000000'B, which is the binary representation of the decimal zero.
point (dot). The routine PUTOUT then uses the vector OUTVEC to create an observation in the SAS dataset. This process is repeated for all rows contained in the ISPF table. The SAS dataset and the ISPF table are then closed, which brings the procedure to an end.

Overview of Application

As mentioned earlier, the use of Proc SPFLINK in the present application allowed us to capitalize on all the advantages of a SAS database, and, at the same time, to create a system for users without extensive computer knowledge. The result is that simple interaction with ISPF panels is all that is necessary for rapid updating of a SAS database.

Under the supervision of the Dialog Manager the procedure invokes the ISPF Services as required. First the main menu is displayed (see Panel 1). When option 1 (‘update task, job ratings’) is chosen, a CLIST allocates the necessary SAS datasets and invokes SAS. Proc SPFLINK then creates the ISPF table.

Once all the observations contained in the SAS dataset have been transferred, a secondary panel, the ‘Update Menu’ is displayed (see Panel 2). Now the user may choose to add, delete, or modify a row in the ISPF table. If the ‘Modify’ option were selected, for example, the changes required will be solicited from the user, then a check will be made as to their appropriateness, and the ISPF table updated (see Panel 3). When all modifications have been made to the table, Proc SPFLINK is called again to write the table back to the SAS dataset.

Conclusion

SPFLINK is a simple to use, inexpensive tool which can increase the accessibility of a SAS database. The procedure as described is specific to the present application, and modifications could allow adaptation to other related problem areas.

For Further Information:

Contact: Luigi Muro
Behavioural Sciences Group
Bell Canada
620 Belmont St., Rm. 1120
Montreal, Quebec, Canada
H3B 2M3
(514) 870-4365
**VARIABLES CONSTITUTE ITS ELEMENTS THE LENGTH OF THE**

**LUNGHT**

**MUST BE**

**IF**

**CALL**

**PROCEDURE**

**PROCEDURE**

**PROCEDURE**

**PROCEDURE**

**PROCEDURE**