Paper 23-25

Let Them Have It Their Way: EZFLXRPT, an Interactive Request Interface for Easy and Flexible Ad Hoc Reporting Based on the SAS® System

LeRoy Bessler
Strong Smart Systems, Bessler Consulting & Research, bessler@execpc.com

Abstract

No, the user does NOT have to make up her/his mind. You can, indeed, build a WRITE-ONCE / USE-MANY / DO-ANY reporting application: build a tool, not a straightjacket. It is possible to provide users at any SAS site with reporting flexibility with minimum software investment.

When a professional programmer designs and builds a reporting application, there is the understandable, but nevertheless unreasonable, expectation that intended users can foresee all needs. Deadly for deadlines, estimates, and budgets is expansion in project scope once work has begun, or is believed complete.

A "no-brainer", fixed-cost-certain way to provide user-definable ad hoc reporting is to buy more software, typically a fancy query or reporting tool. Well, it may come with its own limitations, and the inevitable tormenting temptation of future upgrades, to trade new limitations for old.

This paper discusses at a high level, and the presentation will demonstrate in detail, how much flexibility and usability you can deliver in an ad hoc reporting application that was built with just Base SAS software and its no-added-cost macro facility.

Introduction

Applications designed and built with Software Intelligence (SI) are robust, made of reusable parts, and easy and quick to extend or maintain.

In SUGI papers since 1991, the author has shown how macros, macro variables, parameter files, etc. can be used to build applications that are reliable, reusable, extendable, and maintainable (see Bibliography).

The common internal feature of all SI applications is their dynamically auto-customizing code, for which SAS macro language is the key. EZFLXRPT is likewise built with SAS macro language, and relies on dynamic auto-customization to perform its task.

EZFLXRPT dynamically builds a DATA Step for data extraction, and then PROC SORT and PROC PRINT Steps. The DATA Step and PROC Steps flexibly implement the exact and varying desires of the report requestor, at any point in time, rather than being part of a hard-coded inflexible application that can do only one report with one set of variables.

Of course, something other than PROC PRINT could be used for report functions. EZFLXRPT is not meant at this stage to be the "Be All and End All" tool. Rather, it is a prototype and demonstration of what can be done with basic tools of Base SAS software, when augmented by the power of SAS macro language.

At its current stage of development, EZFLXRPT runs to about 1400 lines (not to be confused with 1400 statements). It is impractical to include in the paper. Anyone interested in the code should communicate with the author.

Genesis

A site had mainframe online TSO SAS applications that had been built many years ago, using Version 5 of Base SAS and SAS/AF®. The applications used early features of SAS/AF that had been incompatible with even the first release of Version 6. The conversion target environment was release 6.08 of SAS software. Furthermore, since no new development was being done with SAS/AF, there was a business decision to convert the applications in such a way that SAS/AF, and SAS/AF expertise, would

not be required. SAS macro language was going to be part of the tool set, regardless.

The applications use the %WINDOW and %DISPLAY macros to provide a non-glitzy, but fully functional, TUI (Textual User Interface). They still run under mainframe TSO, and are accessed via a 3270 emulator package on PCs.

There is no Point and Click. The original applications used SAS/FSP® for query and update, as do the converted applications. In that context, there are optional drop-down lists to get/see valid data selection values. (Also, if an invalid value is entered for certain variables, the appropriate valid list automatically drops down, without explicit request.) SAS/FSP was one of the earliest SAS software components, and is probably under-appreciated. It has no role in EZFLXRPT, however.

The applications had several report functions, which were customizable on a very limited basis by simply filling in the blanks and/or marking with X's on a report-specific submenu. Report formats and options were frozen.

Though one application already had a rather broad collection of custom reports, the other was very light in terms of reporting capability. This is the application the needs of which inspired the concept of EZFLXRPT.

The application's data structure was simple. There was only one SAS dataset. However, it contained dozens of variables, of differing characteristics: two primary keys, several character classification variables, textual description variables, long and short comment variables, lots of date variables, and, of course, numeric variables.

Since the application was on the workbench for conversion anyhow, this was a one-shot opportunity to maximize its hitherto minimal reporting functions. We looked at the question of how to provide ease and flexibility for the users who did not want to be locked into a once and forever definition of their reporting needs. (And there might not be authorization and funding for future enhancements to satisfy needs that are identified later.)

The simple prototype discussed in this paper, EZFLXRPT, is an example of how ease and flexibility objectives can be met. It should be understood that the EZFLXRPT application presented here has no relation to the real business application.

Design and Function

The report request interface uses four screens.

The first two screens are used to define selection parameters for the data.

Screen 1 permits specification of conditions of the form EQ, NE, LE, GE, LT, and GT, where we mean the standard SAS language abbreviations for "conditional connectors" (e.g., "GT" stands for "greater than" or ">"). The user does not code conditional statements, but instead simply enters, e.g., a zero in the appropriate column on the line for a variable that must be greater than zero. The screen layout is six columns for the six possible conditional limits, with a seventh column at left for the list of variables. There are as many rows as there are variables as are eligible for this type of data selection. The width of the cells in any particular row is customized to the maximum width of that variable's values. There is no grid. So there is no visual confusion. Cell width simply manifests itself by how many digits and/or letters the user is permitted to key for the particular variable.

Screen 2 permits specification of a list of one or more permissible selection values for each variable on that screen. Each row has a variable name at left, with a long space at right in which values are entered as a list in single quotes, separated by commas—just like what one would use as the argument of the IN (. . .) condition in the SAS language. We don't think this format is too arduous or difficult to understand. For long text items, the selection

parameter is up to 40 characters, plus 2 quotes. For long text, the 40 characters must be contiguous in the actual variable value, but they need not be the first 40. Also, if needed, the maximum string length could be implemented as 62 (if screen line width is limited to 79), if one gave up the placement of the screen note that stipulates the specific character count limit for that line. Furthermore, if one chose either to use the metadata in the source data library, or to pre-analyze the actual data at start of session to determine the real maximum length of the variable, the data entry line could be dynamically customized to the exact space needs of long text.

When the user leaves either Screen 1 or Screen 2, the entered values are edited for feasibility. E.g., one cannot enter comparison values for both EQ and NE on the same variable. Also, the user can toggle between Screen 1 and Screen 2 to review the full data selection request before launching the data extraction.

The response from data extraction is simple Screen 3. It states how many records (or "observations" or "rows") were selected based on the Data Restrictions entered via Screens 1 and 2. If none were selected, either the desired conditions are not met by the data set, or a data selection error was made. If a huge number of records qualified, the user may want to go back and refine the Data Restrictions, or to try to structure the subsequent report output in such a way that the report, despite its expected size, is easier to use. In any case, whether the record count is zero, or too small, or too large, the user can go back to Screen 1 and/or Screen 2 to revise the Data Restrictions.

If the record count is acceptable, the user simply presses the Enter Key to move on to Screen 4, to specify and submit the actual report request.

Screen 4 lists all the variables in the data set, and provides several columns with which to specify the use, if any, of each in the report. Variables that are assigned either a column number or sort sequence number will appear in the report. In lieu of a column number, it is permissible to enter a "column action". These are response values which trigger appearance of a sorted variable in a BY statement and/or ID statement in the dynamically built PROC PRINT code. Specifying a sorted variable for a Totals Break and/or Page Break triggers its appearance as the argument of a SUMBY and/or PAGEBY statement. Flagging any sorted or ordinary column variable for "Total" triggers its appearance in a SUM statement. Finally, flagging a variable for "Decode" triggers its appearance in a FORMAT statement, which references a SAS format which has a name that is a standard derivative of the variable name. (E.g., variable **tloca** has available format $TLOCAFM.).

Future Enhancements

Implementation of EZFLXRPT requires explicit coding of certain statements, specific to the variables. It is not a dauntingly large number of statements, but it does require hard-coding, and going into and tampering with a big machine, which is known to be reliable as *originally* written. (No change is a guaranteed safe change a priori.)

A better approach would be to use the metadata for the variables that is retained in the SAS data library, to the extent that it is useful, and to supplement that with a simple application dictionary. The dictionary would identify which of the full set of variables are of interest, and would specify which go on Screen 1 versus Screen 2, and whether, for those character variables, if any, which are handled on Screen 2, the data selection should be that for short values or long text. It would also specify which reporting options are permitted for each variable.

When EZFLXRPT is converted to a general-purpose macro, any new or revised application would require merely specifying the input data set and the application dictionary member. Data-selection or report-specification options would be controlled via the dictionary, not by

changing EZFLXRPT internal code. This is a strategy for Reliability, Reusability, Extendability, and Maintainability—which are the previously reported benefits of Software-Intelligent application development.

And, besides the objectives of programmer productivity and effectiveness above, other obvious objectives are user productivity and effectiveness.

The big benefit of a narrowly defined, permanently hard-coded report writer is that the user always gets the right thing with no extra work and no repetition of work done before.

To assure that the user can again reliably get a report previously requested with EZFLXRPT, it would be sufficient to list the data-restriction and report-format specifications as subtitles and/or footnotes on the resulting report. If the list is short, there are presumably enough TITLE and FOOTNOTE statements (10 + 10 in Version 6) available to the PROC PRINT Step. In any case, an extra PROC PRINT Step, either before or after the Step that creates the report, could write the audit trail if preferred, or if necessary due to its length.

Whether such an audit trail is supplied or not, it would be valuable for the user to be able to save the finished DATA Step, PROC SORT, and PROC PRINT code in a recallable and re-executable file. If the user is confident that a saved report definition is acceptable as is, she/he could do a direct submission. However, to revise a pre-existing report, either as a permanent change, or as a new report, the file would be used to pre-load all the screen fields. The user could then work through the screens, and change them where needed.

Bibliography

Among the author's early and more recent papers on Software-Intelligent application development published by SAS Institute Inc. (Cary, N.C.) are: "Intelligent Production Graphic Reporting Applications", in *Proceedings of the Sixteenth Annual SAS Users Group International Conference*, 1991; "Software Intelligence: Applications That Customize Themselves", in *Proceedings of the Eighteenth Annual SAS Users Group International Conference*, 1993; "Reusable, Extendable, Maintainable, Reliable Application Development: Using Software Intelligence to Build an EIS with Only SAS & SAS/GRAPH® Software", in *Proceedings of the Twentieth Annual SAS Users Group International Conference*, 1995; and "Strong Smart Systems: Software-Intelligent Development for Reliable, Reusable, Extendable, Maintainable Applications", in *Proceedings of the Twenty-Fourth Annual SAS Users Group International Conference*, 1999.

Trademarks

Author

LeRoy Bessler, Ph.D.
Strong Smart Systems
Bessler Consulting & Research
PO Box 96
Milwaukee, WI 53201-0096, USA
bessler@execpc.com
414-351-6748

LeRoy Bessler is a SAS consultant, with interests in macro-based Software-Intelligent Application Development, visual communication, graphic design, information visualization, color, and InfoGeographics. An internationally recognized expert on SAS/GRAPH and graphic design, and an award winner for papers on graphic design and visual communication, Dr. Bessler is writing a book titled "Chart Smart: Design Guide and Solution Toolkit for SAS Graphs, Tables, and Maps That Inform and Influence".