Reuseable, Extendable, Maintainable, Reliable Application Development: Using Software Intelligence to Build an EIS with Only SAS & SAS/GRAPH Software
LeRoy Bessler, Miller Brewing Company

Introduction

How the objectives of reusability, extendability, maintainability, and reliability can be met with Base SAS* and SAS/GRAPH* software is discussed, and illustrated with a mini-prototype EIS (Enterprise Information System) developed using Software Intelligence (SI). Parameter files, SAS macros, and macro variables are the enablers of SI—used to build applications that do dynamic autocustomization (i.e., applications that modify themselves).

Robust, easy-to-use applications for users, and techniques to maximize developer productivity, need no special software, fancy interfaces, or new nomenclature. Macros are my favorite "objects".

The %WINDOW and %DISPLAY macros in Base SAS can meet online application user interface needs. Base SAS and SAS/GRAPH can meet all data presentation needs for an EIS. Base SAS can meet all common data analysis needs.

A concise, easy-to-use TUI (Textual User Interface) supports "Single-finger Computing"—within the capability of any user who can read. Icon clutter and a mouse are unnecessary. Automated teller machines, on-screen programmable VCRs, and walk-up interactive information kiosks—all relying only on text video display and a keypad, keyboard, or touchscreen "virtual keyboard"—serve millions of computer-naive people every day. Why should computer users in business, government, nonprofit organizations, or academe have to settle for less?

The EIS prototype was developed without the extra knowledge, manuals, and expense of another language (Screen Control Language, SCL).

The EIS permits menu-based, fill-in-the-blanks specification of print destination ID, and selection of "canned" graphs for display or print. (It is a prototype in that it offers only a single menu of graph selections, rather than a nest of menus and other information delivery functions.)

By "canned" graphs is meant that design and content, but not data, for the graphs are predefined. Graph design remains unchanged from run to run, but not the data. For methods to dynamically autocustomize the details of an application from run to run—to suit the vicissitudes of data and/or date while meeting graph or report design requirements—see my earlier paper "Software Intelligence: Applications That Customize Themselves", on pages 114-122 of Proceedings of the Eighteenth Annual SAS Users Group International Conference, SAS Institute Inc. (Cary, N.C.), 1993.

In the present paper, however, no details of the actual graph programs are given. Those application-specific programs are in a library referenced by the EIS.

Complete code for the EIS prototype is furnished. It can readily be adapted: alter the graph selection list, point the EIS at your graph programs, and customize the print-support code for your device and network.

In the Appendix are the TSO CLIST to invoke the application, the AUTOEXEC file that is one of the parameter files, the program code, the general-use macros in an AUTOCALL library, and exhibits of the Printer ID and Graph Selection windows. During the presentation of this paper, sequences of screen prints demonstrate use and operation of the EIS, and the effect of the controlling program statements.

This paper assumes an understanding of SAS macro processing in general, and merely presents, rather than explains, use of the %DISPLAY and %WINDOW macros. (All items in the Appendix are commented, rather than explained in the body of the paper.) For more information, please see SAS Guide to Macro Processing, SAS Language: Reference, SAS Language and Procedures: Usage 2, and SAS Companion for the MVS Environment, all published by SAS Institute Inc.

Software Intelligence

First, let me acknowledge that SI is really a collection of old techniques—not invented by me, and not recently invented by anyone else—for building what I call bullet-proof, hands-off applications.

But, because many people in the SAS user community are not veteran professional application developers, SI is a way of working that I feel is important to advocate and to share. SI is necessary for Maturation in SAS System Use.

The first stage of SAS System use is as an end-user tool—for ad hoc data analysis/presentation tasks. The second stage is when the site's SAS support staff enhance SAS software as an enterprise-wide utility, by providing site-specific customization, macros, formats, templates, etc. The final stage is SAS software as a production application development tool, whether for scheduled (possibly computer-scheduled) batch processing, or for an on-demand online EIS.
Ad hoc processing is typically: (a) one-time or irregularly needed; (b) custom and iterative in development of the program code; and (c) often done interactively. Production processing is standardized and hands-off, and must get everything right the first time, every time.

Software Intelligence permits an application program to dynamically customize itself, without direct human intervention, to continue to meet design objectives in a changing environment.

Reliability

"Reusability, Extendability, Maintainability, and Reliability . . . the greatest of these is Reliability."

A malfunctioning application is worthless.

The key to reliability is simple--once your program is working right, never touch it again.

The only safe program change is no change.

More hazardous than changing your own long-in-service program is to change one that someone else wrote. Most hazardous is to change a program that several people have maintained. Part of an old program may even be doing no longer needed processing and producing no longer referenced outputs. Such refusal or neglect to maintain the program is a tacit, pragmatic admission that no change is a safe change.

But, since user needs do change, an application program must change to meet those needs. What's a reasonable middle course?

Any foreseeable change is best supported through Software Intelligence.

For example, if a tabular or graphic report takes as input the last N years, months, weeks, or days of historical data, it may be prudent to keep N in a parameter file that is read by the program, rather than "hard code" N in the program itself. Every time you or a successor might open the program to change N, the program would be at risk.

Another good candidate for storing in a parameter file is a goal or threshold for a measurement variable. Since management's judgement of what is good or bad changes over time, plan ahead to accommodate that without program change.

Common for novice application developers, especially computer end users as opposed to IS professionals, is to include card data inside the program. This can be an expedient productivity aid during application development and debugging, when you don't want to have to go to a separate file to change the input every time you need to test a different case. However, when development is complete, separate the data from the program.

Too frequent in applications, especially if originally written for a supposed one-time analysis or report (any ad-hoc application--if valuable--is likely to experience recurrent use), are manually entered dates, for a title and/or for a filter on data selection. If the date is dependably a function of run-date, let the program use SAS functions to retrieve today's date and to compute and construct the title or filter date(s) from it. If not a function of run-date, supply the manually entered date via a parameter file, and let the program read it from there.

Every time you or a successor might open a program to change a reporting parameter like N (see above), to change a goal or threshold, to change card data, or to change a date, the program would be at risk.

Program-change avoidance (i.e., reliability enhancement) is implemented in the examples above by building "Fire Walls": Build fire walls between your program and your data, between your program and your (or a successor's) keyboard.

By use of parameter files, macros, and macro variables, SI can provide an automated change interface--to protect program integrity, but still support limited revision of format, content, or function (to support a "flexible freeze").

The Reality of Maintainability

No one who advocates a real or illusory change in technique for application development ever dares to try to convert the programming community without self-serving, often statistically "proven", claims of easier maintainability. Here, instead, are--

- **Bessler's First Theorem**: The only easily maintained application is one for which maintainer and creator are the same person.

  Corollary: Ease of maintenance is inversely proportional to k raised to the power N-1, where N is the number of distinct persons who have touched it. (Count the creator when computing N.) k is a constant, greater than 1. Its exact value still needs to be discovered.

- **Bessler's Last Theorem**: The only very easily maintained application is one which the maintainer originally developed recently--within the last few weeks, preferably yesterday.

  This is Bessler's Last Theorem on application maintainability, not Bessler's last theorem on any topic ever.

Software Intelligence, as used in this paper and elsewhere, makes program and macro maintenance infrequent. Only an application design change would require a code rewrite.
Reusability

The old concept of reusable components for application development was not invented by newcomers to the programming field. Reusability has nothing intrinsically to do with so-called "objects", Object-Oriented Programming (OOP), or any specialist nomenclature that defines OOP ideological purity.

In the context of SAS software, reusability is enabled through either includable blocks of source code, or SAS macros. The code or macros are best stored in shared-access libraries so that anyone—with documentation as to availability, purpose, required inputs, and provided outputs—can use them.

Unfortunately, including reusable code by saying "%INCLUDE sourcefilename" fails to disclose—in that statement—what the inputs and outputs are.

Invocation of a well-designed macro, however, explicitly identifies the names of the inputs and outputs via assignment of values to parameters. Other parameters may be used to control the function of the macro. So, macro use not only documents the program, but also reduces the likelihood of error in use of the reusable entity.

The macros %GETTTLS and %SLCTLNS are not designed for general use. They are coded "in-stream" in the %EISPROTO macro, not stored in an AUTOCALL library.

Extendability ("Extensibility" Is Not a Word)

When I wrote the first edition of the EIS prototype, every time the number of graph selections on the menu changed (typically, increased) I had to change lots of program code.

Now the number of graph selections is controlled via a macro parameter, supplied "outside" from the AUTOEXEC file. This provides extendability (or shrinkability) by requiring the change of only one number, and protects the working program code.

The benefit is non-trivial since—prior to the extendable macro implementation—each selection line required its own screen definition code, its own response-field initialization code, its own response editing code, its own reaction-to-response code, etc.

The macro’s Software-Intelligent design enables dynamic autocustomization of the application, with no code change by the application support person every time user needs change.

Accelerating Application Performance

If graph creation requires a lot of up-front "number-crunching", that can have a big impact on resources and performance. The solution is to do number-crunching in a preliminary batch job (rerun only based on the reporting cycle or data-refresh cycle, whichever is longer), to create the final input files for the online graph programs. This is like client/server processing, but with a static batch-updated server and dynamic online clients.

Notices

The EIS prototype was developed and tested with SAS Release 6.08 on MVS TSO. I believe it to be reliable, but can offer it only "as is". If adapted by you, the prototype must be adjusted by you for your environment, and must be tested by you to verify acceptability and correctness of results.

SAS and SAS/GRAPH are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. * denotes USA registration.

Author

Dr. LeRoy Bessler
P.O. Box 96
Fox Point, WI 53201-0096, USA
Telephone: 414-351-6748

Appendix

EIS TSO CLIST

%SAS AUTOEXEC(''SITE.SASPGLMS(EISINIT)'') +
   INPUT(''SITE.SASPGLMS(EISMAIN)'') SASAUTOS(''SITE.USERMACS''') +
   LOG(1) UNITS(CYLINDERS) /* suppresses the SAS log */

EIS Program Initialization Code (i.e., AUTOEXEC for CLIST)

FILENAME DPRINTID 'EIS.PRTDEFIL'; /* must be only 1 observation */
FILENAME PROGRAMS 'EIS.GRAPHLIB';
%LET HOWMANY=17; /* GRAPHLIB must have members GRAPH1 through GRAPHN, */
RUN; /* where N (in this case) must be 17 */
EIS Main Program Code (i.e., INPUT for CLIST)

OPTIONS CMDMAC;

/* block Q (Query) command entered at COMMAND line ******************* */

    /* to block more commands, */
    %MACRO Q / CMD; /* repeat this block of statements */
    END /* (%MACRO through %MEND), and */
    %MEND Q; /* replace 'Q' with command to block */
    /* *********************************************** */

%EISPROTO(COUNT=&HOWMANY)
RUN;

General-use Macros in Shared-access AUTOCALL Library
(Note that the %EISPROTO macro contains two in-stream macros)

%MACRO VARLIST(PREFIX=,VARCOUNT=);
%DO I = 1 TO &VARCOUNT; /* create list of same-purpose variables, */
      &PREFIX&I /* all with same prefix, */
%END; /* but with incremented sequence number */
%MEND;

%MACRO EISPROTO(COUNT=25); /* ****** start of EISPROTO macro ****** */
/* ********** pass default Printer ID as global variable ********** */
DATA _NULL_; INFILE DPRINTID; INPUT @1 PRINTER $4.; %GLOBAL DEFAULTI; CALL SYMPUT('DEFAULTI',PRINTER); RUN;
/* *********************************************** */
/* ********** define the Printer ID window *************** */
%WINDOW PRINTER
#01 32 "Default printer ID is"
     324 DEFID PROTECT=YES
#03 32 "To override, type another ID of form U9nn"
#04 32 "where nn is appropriate two digits"
#06 32 "Desired printer ID =>"
     324 PRTID 4 REQUIRED=NO AUTOSKIP=YES
#08 32 "Press Enter to continue";
/* *********************************************** */
/* ********** permit override of default Printer ID ********** */

%PRINTER: /* just a label */
%LET DEFID = &DEFAULTI; /* load default printer ID */
%LET PRTID = ; /* initialize data-entry variable */
%DISPLAY PRINTER; /* display Printer ID window */
%IF &SYSCMD NE %THEN %DO;
  %LET SYMSG = Invalid Use of COMMAND===> line;
  %GOTO PRINTER;
%END;

/* below, AFPPREP is site-specific macro for printer setup */
%IF &PRTID EQ /* if no override, */
  %THEN %AFPPREP(AFDEST=&DEFAULTI); /* then do setup with default */
%ELSE %DO;
  %LOCAL MPRTID;
  %LET MPRTID = %UPCASEC&PRTID); /* translate to upper case */
  %IF %SUBSTRC&MPRTID,l,l) NE U
    OR %SUBSTRC&MPRTID,2,1) NE 9
    OR %SUBSTRC&MPRTID,3,1) LT 0
    OR %SUBSTRC&MPRTID,4,1) LT 0
    %THEN %DO;
      /* else edit */
      /* translate to upper case */
      %LET SYSMSG = ID must be of form U9nn; /* set error message */
      %GOTO PRINTER; /* re-display Printer ID window*/
      %END;
    %ELSE %AFPPREP(AFDEST=&MPRTID); /* OK, do setup with override */
    %END;
/* *************************************************************** */
/* ********** define the Graph Selection window ****************** */
%MACRO GETTTLS; /* IN-STREAM MACRO for getting graph titles */
%DO I = 1 %TO &COUNT;
  %GLOBAL TITLE&I;
  INFILE PROGRAMS(GRAPH&I);
  INPUT &I LINE $80.;
  IF SUBSTR(LINE,1,7) = 'TITLEI ';
  PRESTART = INDEX(LINE,'''');
  LINEEND = SUBSTR(LINE,PRESTART+1,78-PRESTART);
  AFTEREND = INDEX(LINEEND,'''');
  TITLE = SUBSTR(LINEEND,1,AFTEREND-1);
  CALL SYMPUT("TITLE&I",TITLE); RUN;
%IF &SYSERR > 0 %THEN %DO; /* handle missing program situation */
  %WINDOW ERROR
  #01 &2 "Error encountered - processing cannot continue"
  #02 &2 "Tell EIS Administrator program GRAPH&I is missing"
  #04 &2 "Press Enter to exit";
  %DISPLAY ERROR;
  ENDSAS; /* end the EIS session */
%END;
%MEND GETTTLS;

%GETTTLS
RUN;
MACRO SLCTLNS; /* IN-STREAM MACRO for graph-selection screen lines */
LOCAL J;

DO I = 1 TO &COUNT;
  LET J = EVAL(I + 3);
  &J 2 SELECT&I 1 REQUIRED=NO
  AUTOSKIP=YES @4 "&TITLE&I"
END;
%END SLCTLNS;

WINDOW SELGRAPH /* define the window */
01 @2 "Select a Graph: Type D for Display, P for Print, Q for Quit"
02 @2 "Then press Enter"
%SLCTLNS
; /* this semi-colon terminates %WINDOW statement */
/* *************************************************************** */
/* ********** get the user's Graph Selection ********************* */
SELGRAPH: /* just a label */
LOCAL %VARLIST(PREFIX=SELECT,VARCOUNT=&COUNT);

DO K = 1 TO &COUNT;
  LET SELECT&K = ; /* initialize data-entry variables */
END;

DISPLAY SELGRAPH;

IF &SYSCMD NE %THEN DO;
  LET SYMSG = Invalid Use of COMMAND==> line;
  GOTO SELGRAPH;
END;

LOCAL %VARLIST(PREFIX=MSELEC,VARCOUNT=&COUNT);

DO L = 1 TO &COUNT;
  LET MSELEC&L = UPCASE(&SELECT&L);
END; /* convert data-entry variables to all upper case */

LOCAL COUNTSEL;
LET COUNTSEL = 0;

DO M = 1 TO &COUNT;
  IF &MSELEC&M NE %THEN DO;
    IF &MSELEC&M EQ Q
      THEN GOTO EXIT; /* user said 'Quit' */
    IF &MSELEC&M NE D
      AND &MSELEC&M NE P %THEN DO;
      LET SYMSG = Selection Code invalid;
      GOTO SELGRAPH;
    END;
  ELSE DO; /* selection response was valid */
    LOCAL DEVTPE;
%LET DEVTYPE = &&MSELEC&M;
%LOCAL GRAPHID;
%LET GRAPHID = &M;
%END; /* since it was valid, now count it */
%LET COUNTSEL = %EVAL(&COUNTSEL + 1);
%END;
%END;
%IF &COUNTSEL NE 1
%THEN %DO; /* handle the error */
%IF &COUNTSEL EQ 0
  %THEN %LET SYSMSG = Type a Selection Code;
%ELSE %LET SYSMSG = Select only One Graph;
%GOTO SELGRAPH;
%END;
/* *************************************************************** */
/* If not sent back to SELGRAPH, continue from here. Specify GOPTIONS & run the selected graph program */
%IF &DEVTYPE = D %THEN %DO;
  GOPTIONS DEVICE=DISPLAY;
%END;
%ELSE %DO;
  GOPTIONS DEVICE=PRINTER;
%END;
/* *************************************************************** */
%INCLUDE PROGRAMS(GRAPH&GRAPHID);
RUN; /* make sure no RUN after graph PROC in the included program -- for some PROCs two RUN statements will produce same graph twice */
/* *************************************************************** */
/* ********** user said 'Quit' ****************************************************** */
%IF &DEVTYPE = P
  %THEN %LET SYSMSG = Graph Created for Printer; /* Reassure user that something actually happened. Display would have been seen, but not so if processing a print request. In case print is created, but not released till end of session, this message could explain. */
/* *************************************************************** */
/* request was handled, now re-display the Graph Selection window */
%GOTO SELGRAPH;
/* *************************************************************** */
/* ********** user said 'Quit' ********************************************************** */
%EXIT:
QUIT;
/* *************************************************************** */
%MEND EISPROTO; /* **** end of EISPROTO macro **** */
Rough Illustration of EIS Windows (not actual screen prints)

PRINTER

Command ==> 

Default printer ID is U987

To override, type another ID of form U9nn
where nn is appropriate two digits

Desired printer ID => _

Press Enter to continue

SELGRAPH

Command ==> 

Select a Graph: Type D for Display, P for Print, Q for Quit
Then press Enter

- Graph 1 Title as specified on its program TITLE1 statement
- Graph 2 Title
- Graph 3 Title
- Graph 4 Title
- Graph 5 Title
- Graph 6 Title
- Graph 7 Title
- Graph 8 Title
- Graph 9 Title
- Graph 10 Title
- Graph 11 Title
- Graph 12 Title
- Graph 13 Title
- Graph 14 Title
- Graph 15 Title
- Graph 16 Title
- Graph 17 Title