Paper 13

Strong Smart Systems: Software-Intelligent Development for Reliable, Reusable, Extendable, Maintainable Applications

LeRoy Bessler
Strong Smart Systems, Bessler Consulting & Research, Fox Point, WI

Abstract

Applications designed and built with Software Intelligence (SI) are robust, made of reusable parts, and easy and quick to extend or maintain. With dynamically auto-customizing code, "living" applications go beyond change tolerance to change amenability, and further—to change implementation. They continue to meet needs in an environment of ever-changing user or management preferences, run-dates, data dates, and data content. Limited changes in report/graph content are handled without reprogramming. This paper provides application components of wide, general use. The code is suitable for immediate use by even a novice or casual user, and illustrates principles and techniques that the very experienced programmer may also find useful.

Introduction

Software Intelligence permits application programs to dynamically customize themselves, without human intervention, to continue to meet design requirements in a changing environment. Rather than static expressions and engines of single-point-in-time programmer decision or user choice, such adaptive programs are "Living Applications". They go beyond change amenability (maintainability) to change auto-implementation. Short of a revolution in host computer "ecology" or in business or research process to be served, such applications have the gift of "Eternal Life".

This paper explains how the objectives of reliability, reusability, extendability, and maintainability can be met with the SAS* System by using Software Intelligence (SI), and illustrates that with examples. Parameter files, macro variables, and macros are SI enablers, for applications that do dynamic autocustomization (i.e., that modify themselves).

Robust applications for users, and techniques to maximize developer productivity, need no special software or new nomenclature. Macros are my objects.

My past implementations of dynamically autocustomized applications—that can handle, from run to run, the vicissitudes of data and/or date while meeting graph or report design requirements, and that adapt to changing user or management preferences—were documented in a series of papers. (See the bibliography.)

This paper assumes an understanding of SAS macros.

Software Intelligence

SI is really a collection of old techniques, not invented by me, and not recently invented by anyone else, for building bulletproof, hands-off applications.

Though not new, SI techniques are a way of working that is important to advocate and to share. Many people in the SAS user community are not long-time professional application developers. 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 or data 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 as a production application development tool, whether for on-demand online or scheduled (possibly computer-scheduled) batch processing.

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, online or batch, is standardized and hands-off. It must get everything right the first time, every time.

Reliability

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

A malfunctioning application is an impediment to productivity and a barrier between you and the information you need.

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 admission that no change is a safe change.

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

Foreseeable change can be, and 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 is more 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 to store in a parameter file is a goal or threshold for a measurement variable. Since judgement of what is good or bad changes over time, it's best to 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 data in the program. This is a productivity aid during application development and debugging, when you don't want to go to a separate file to change the input every time you need to test a different case. However, when development is complete, data should be separated 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 very likely to experience recurrent use), are manually entered dates, for a title and/or for a filter on data selection. If such a 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, for the program to read.

Program-change avoidance (i.e., reliability enhancement) is implemented in the examples described above by what I call "Building Firewalls". **Build Firewalls** between your program and the data, between your program and your (and everyone else's) keyboard.

With parameter files, macro variables, and macros, SI can protect program integrity, but still support limited revision of format, content, or function—to support a "flexible freeze" (to borrow a phrase from the 1970's USA-USSR arms control dialogue).

Reusability

The concept of reusable components is not new. Reusability has nothing intrinsically to do with "objects". In the context of SAS software, reusability is enabled through either includable blocks of source code, or macros. Code or macros are best stored in shared-access libraries so anyone, furnished with documentation as to availability, purpose, required inputs, and provided outputs, can use it or them.

Unfortunately, including reusable code by saying "%INCLUDE *sourcefilename*" fails to disclose what its inputs and outputs are.

Invocation of a well-designed macro, however, explicitly requires identification of the names of the inputs and outputs via assignment of values to parameters. Other parameters are used to control the function of the macro. Such a macro "documents" the program, and is less likely to be erroneously invoked when reused.

Extendability

When I wrote the first edition of a Visual Information System prototype, every time the number of graph selections on the menu changed (typically, increased), I had to change lots of program code. Eventually, I restructured the application with macro processing, and controlled the number of graph selections via a macro parameter, supplied "outside" from a SAS AUTOEXEC file. This provided extendability (or shrinkability) by requiring the change of only one number, and protected the working program code.

The benefit was non-trivial. Prior to the extendable macro implementation, each selection line required its own screen definition code, its own response-field initialization and editing code, etc.

The macro's Software-Intelligent design dynamically autocustomized the application, without reprogramming every time user needs changed.

Maintainability vs. Reality

Advocates of change in technique for application development try to convert the programming community with claims of easier maintainability. Here is the reality.

Bessler's First Theorem: Application maintenance is easy only when 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 persons who have touched it. Count the creator in N. k is a constant, greater than 1. Its exact value still needs to be discovered.

Bessler's Second Theorem: Application maintenance is very easy only if the maintainer created it recently—within the last few weeks, preferably yesterday.

Software Intelligence can make application maintenance rare, quick, and safe. With SI design, all foreseeable changes can be delivered by merely updating one or more parameter files, rather than directly changing the macro parameters in static program code.

Examples

To respect page count limits, and to avoid spending space on all the details of a big application, most examples in the Appendix are only components, which may be widely usable, or are little applications. Their use and function are explained there.

Related SUGI Papers by the Author

Published by SAS Institute Inc. (Cary, NC) 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; and "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.

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

Dr. LeRoy Bessler is a SAS consultant, and frequent speaker, with interests in Software-Intelligent Application Development, visual communication, graphic design, information visualization, color, and InfoGeographics. An award winner for papers on graphic design and visual communication, he is writing a book to be published by SAS Institute, titled "Chart Smart: Design Guide and Solution Toolkit for SAS Graphs, Tables, and Maps That Inform and Influence".

Appendix: Examples of Software-Intelligent Application Development

Example 1.

```
DATA TESTREF; /* included for illustration only */
INFILE CARDS; /* data should be kept separate from program */
INPUT REFIN;
CARDS;
100
 ;
RUN;

%MACRO GETREF(EXTDATA=,IREFVAR=,OREFVAL=);
DATA _NULL_;
SET &EXTDATA;
%GLOBAL &OREFVAL;
CALL SYMPUT("&OREFVAL",TRIM(LEFT(&IREFVAR)));
RUN;
%MEND;

%GETREF(EXTDATA=TESTREF,IREFVAR=REFIN,OREFVAL=MINVALUE)
RUN;

DATA EXAMPLE1; /* included for illustration only */
INFILE CARDS;  /* data should be kept separate from program */
INPUT TESTVALU;
IF TESTVALU GE &MINVALUE;
CARDS;
1
20
300
4000
500
60
7
 ;
RUN;

OPTIONS NODATE NONUMBER;
TITLE "Listing of Observations with TESTVALU Not Less Than &MINVALUE";
FOOTNOTE;
PROC PRINT DATA=EXAMPLE1 NOOBS;
VAR TESTVALU;
RUN;
```

Output from PROC PRINT:

Listing of Observations with TESTVALU Not Less Than 100

TESTVALU
300
4000
500

Example 2. Extract from a data set the earliest and latest date keys, and supply them as formatted global variables for use in a TITLE statement.

```
%MACRO DATERANG(DATA=,DATEVAR=,FDATEVAR=,LDATEVAR=);
PROC SORT DATA=&DATA OUT=DATES(KEEP=&DATEVAR) NODUPKEYS;
BY &DATEVAR;
RUN;
DATA _NULL_;
%GLOBAL &FDATEVAR &LDATEVAR;
SET DATES END=LAST;
BY &DATEVAR;
IF _N_ = 1 THEN DO;
  MONTEXT  = PUT(&DATEVAR,MONNAME9.);
  DAYTEXT  = PUT(&DATEVAR,DAY2.);
  YRTEXT   = PUT(&DATEVAR,YEAR4.);
  DATETEXT = PUT((TRIM(LEFT(MONTEXT))||' '||
                 TRIM(LEFT(DAYTEXT))||', '||TRIM(LEFT(YRTEXT))),$18.);
  CALL SYMPUT("&FDATEVAR",TRIM(LEFT(DATETEXT)));
END;
ELSE IF LAST THEN DO;
  MONTEXT  = PUT(&DATEVAR,MONNAME9.);
  DAYTEXT  = PUT(&DATEVAR,DAY2.);
  YRTEXT   = PUT(&DATEVAR,YEAR4.);
  DATETEXT = PUT((TRIM(LEFT(MONTEXT))||' '||
                 TRIM(LEFT(DAYTEXT))||', '||TRIM(LEFT(YRTEXT))),$18.);
  CALL SYMPUT("&LDATEVAR",TRIM(LEFT(DATETEXT)));
END;
RUN;
%MEND  DATERANG;

DATA EXAMPLE2; /* included for illustration only */
INFILE CARDS;  /* data should be kept separate from program */
INPUT @1 DATADATE YYMMDD10. @12 DATAVALU $40.;
CARDS;
. . .
;
RUN;

%DATERANG(DATA=EXAMPLE2,DATEVAR=DATADATE,FDATEVAR=FRSTDATE,LDATEVAR=LASTDATE)
RUN;

OPTIONS NODATE NONUMBER;
TITLE "Unsorted Observations from &FRSTDATE to &LASTDATE";
FOOTNOTE;
PROC PRINT DATA=EXAMPLE2 NOOBS;
FORMAT DATADATE YYMMDD10.;
RUN;
```

Output from PROC PRINT:

```
 Unsorted Observations from July 11, 1917 to December 4, 1997

             DATADATE    DATAVALU

           1945-09-04    Birth of Carol
           1917-07-11    Birth of Chester
           1919-09-13    Birth of Martha
           1940-07-20    Wedding of Martha & Chester
           1963-02-02    Wedding of Carol & LeRoy
           1949-09-11    Birth of Gerald
           1941-12-13    Birth of LeRoy
           1997-12-04    Today's WISAS Meeting
```

Example 3. Extract from a data set the observation(s) for a selected date, and supply the date as a formatted global variable for use in a TITLE statement.

```
%MACRO SLCTDATE(SLCTYR=,SLCTMO=,SLCTDA=,CHKDATE=);
%GLOBAL &CHKDATE;
%IF &SLCTYR NE AND &SLCTMO NE AND &SLCTDA NE %THEN
  %LET &CHKDATE = %SYSFUNC(MDY(&SLCTMO,&SLCTDA,&SLCTYR));
%ELSE
  %LET &CHKDATE = %EVAL(%SYSFUNC(TODAY()) - 1);
%MEND  SLCTDATE;
%MACRO TTLDATE(INDATE=,
               OUTDATE=);
%GLOBAL &OUTDATE;
DATA _NULL_;
MONTEXT  = PUT(&INDATE,MONNAME9.);
DAYTEXT  = PUT(&INDATE,DAY2.);
YRTEXT   = PUT(&INDATE,YEAR4.);
DATETEXT = PUT((TRIM(LEFT(MONTEXT))||' ' ||
                TRIM(LEFT(DAYTEXT))||', '||
                TRIM(LEFT(YRTEXT))),$18.);
CALL SYMPUT("&OUTDATE",TRIM(LEFT(DATETEXT)));
RUN;
%MEND  TTLDATE;
DATA EXAMPLE3; /* included for illustration only */
INFILE CARDS;  /* data should be kept separate from program */
INPUT @1 EVENTDAY YYMMDD10. @12 EVENT $40.;
CARDS;
1945/09/04 Birth of Carol
1917/07/11 Birth of Chester
1919/09/13 Birth of Martha
1940/07/20 Wedding of Martha & Chester
1963/02/02 Wedding of Carol & LeRoy
1949/09/11 Birth of Gerald
1941/12/13 Birth of LeRoy
1997/12/04 Today's WISAS Meeting
 ;
RUN;
%SLCTDATE(SLCTYR=1917,SLCTMO=7,SLCTDA=11,CHKDATE=SASDAY)
RUN;
%TTLDATE(INDATE=&SASDAY,OUTDATE=DAYTEXT)
RUN;
OPTIONS NODATE NONUMBER;
TITLE "What happened on &DAYTEXT?";
FOOTNOTE;
PROC PRINT DATA=EXAMPLE3(WHERE=(EVENTDAY=&SASDAY)) NOOBS;
VAR EVENT;
RUN;
```

Output from PROC PRINT:

```
              What happened on July 11, 1917?

                          EVENT

                   Birth of Chester
```

Example 4. Rank the Top NN observations from a data set, with NN being selectable and appearing automatically in the title. If the ranking includes all the observations, or if a minimum cutoff is used, then the title says "Ranked List of", rather than "Top NN". A subtitle shows what percent of the total is accounted for in the table. An extra subtitle is generated if the minimum cutoff is used with effect.

Table A: Top 10 SAS PROCs Used From 12-01-93 To 01-31-95

This list accounts for 89.9% of the total Count

Rank	PROC	Count
1	DATASTEP	212,421
2	SORT	70,216
3	PRINT	26,836
4	GPLOT	23,504
5	MEANS	19,103
6	FORMAT	17,522
7	REG	15,047
8	PRINTTO	11,409
9	DATASETS	9,254
10	CONTENTS	7,545
		=======
		412,857

Program A:
```
%TOPNN(DATA=INDATA,
CLASSVAR=SASPROC,
CVARLABL=PROC,
RANKVAR=COUNT,
RVARFMT=COMMA7.,
RVARLABL=Count,
NN=10,
TTLTEXT=SAS PROCs Used From 12-01-93 To 01-31-95)
RUN;
```

Table B: Ranked List of SAS PROCs Used From 12-01-93 To 01-31-95

This list accounts for 86.3% of the total Count

Only values not less than 10,000 are listed

Rank	PROC	Count
1	DATASTEP	212,421
2	SORT	70,216
3	PRINT	26,836
4	GPLOT	23,504
5	MEANS	19,103
6	FORMAT	17,522
7	REG	15,047
8	PRINTTO	11,409
		=======
		396,058

Program B:
```
%TOPNN(DATA=INDATA,
CLASSVAR=SASPROC,
CVARLABL=PROC,
RANKVAR=COUNT,
RVARFMT=COMMA7.,
RVARLABL=Count,
NN=10,
MINRVAR=10000,
TTLTEXT=SAS PROCs Used From 12-01-93 To 01-31-95)
RUN;
```

Macro for Tables A & B:

```
%MACRO TOPNN
(DATA=,CLASSVAR=,CVARLABL=,RANKVAR=,RVARFMT=,RVARLABL=,NN=,MINRVAR=.,TTLTEXT=);
SET &DATA;
%GLOBAL BELOWMIN;
IF _N_ = 1 THEN CALL SYMPUT('BELOWMIN','N');
IF &MINRVAR NE . THEN DO;
  IF &RANKVAR GE &MINRVAR THEN RETURN;
  ELSE DO;
    CALL SYMPUT('BELOWMIN','Y');
    DELETE;
  END;
END;
KEEP &CLASSVAR &RANKVAR;
RUN;
PROC SORT OUT=FORTOPNN; BY DESCENDING &RANKVAR;
RUN;
DATA TOREPORT;
SET FORTOPNN;
IF _N_ < &NN + 1;
RANK = _N_;
RUN;
PROC MEANS DATA=&DATA   NOPRINT SUM N;
VAR &RANKVAR;
OUTPUT OUT=ALL   SUM=SUMTOT N=NTOT;
RUN;
PROC MEANS DATA=TOREPORT NOPRINT SUM N;
VAR &RANKVAR;
OUTPUT OUT=TOPNN SUM=SUMTOP N=NTOP;
RUN;
DATA _NULL_;
MERGE ALL TOPNN;
FORMAT PCTTOT 5.1;
PCTTOT = ROUND((100 * (SUMTOP / SUMTOT)),.1);
%GLOBAL RANKLEN;
CALL SYMPUT('RANKLEN',LENGTH(LEFT(&NN)));
%GLOBAL MIN;
IF &MINRVAR NE . THEN CALL SYMPUT('MIN',TRIM(LEFT(PUT(&MINRVAR,&RVARFMT))));
%GLOBAL PCTTOT;
CALL SYMPUT('PCTTOT',TRIM(LEFT(PCTTOT)));
%GLOBAL HTTLMIN;
FORMAT TTLTOPNN $14.;
IF NTOP < NTOT AND NTOP = &NN THEN DO;
    TTLTOPNN = "Top &NN";
    CALL SYMPUT('HTTLMIN','0');
  END;
ELSE DO;
  TTLTOPNN = 'Ranked List of';
  IF &MINRVAR = . OR "&BELOWMIN" = 'N'
    THEN CALL SYMPUT('HTTLMIN','0');
  ELSE   CALL SYMPUT('HTTLMIN','1');
END;
%GLOBAL TTLTOPNN;
CALL SYMPUT('TTLTOPNN',TRIM(TTLTOPNN));
RUN;
OPTIONS NODATE NONUMBER;
PROC PRINT DATA=TOREPORT NOOBS U LABEL SPLIT='*';
FORMAT RANK     &RANKLEN..;
FORMAT &RANKVAR &RVARFMT;
LABEL RANK      = 'Rank'
      &CLASSVAR = "&CVARLABL"
      &RANKVAR  = "&RVARLABL";
VAR RANK &CLASSVAR &RANKVAR;
SUM &RANKVAR;
TITLE1 "&TTLTOPNN &TTLTEXT";
TITLE3 "This list accounts for &PCTTOT% of the total &RVARLABL";
%IF &HTTLMIN = 1 %THEN %DO;
TITLE5 "Only values not less than &MIN are listed";
%END;
%MEND TOPNN;
```

Example 5. Build a vary-able size menu of graph program selections, using the actual graph title text that is in the graph programs themselves. Programs are kept in an MVS Partitioned Data Set, for which the SAS FILEREF is "PROGRAMS". Each program name is of the form "GRAPHnn", with "nn" being a two-digit number (00, 01, 02, etc.). The maximum value of "nn" is kept in an external parameter file, for which the SAS FILEREF is "PARMFILE". Graph TITLE statements must be of the form "TITLE1 'text';".

```
DATA _NULL_;
INFILE PARMFILE;
INPUT @1 HOWMANY 2.;
%GLOBAL COUNT;
CALL SYMPUT('COUNT',HOWMANY);
RUN;

%MACRO GETTTLS; /* MACRO for getting graph titles */
%DO I = 1 %TO &COUNT;
     DATA _NULL_;
     %GLOBAL TITLE&I;
     INFILE PROGRAMS(GRAPH&I);
     INPUT @1 LINE $80.;
     IF SUBSTR(LINE,1,7) = 'TITLE1 ';
     PRESTART = INDEX(LINE,'''');
     LINEEND = SUBSTR(LINE,PRESTART+1,80-PRESTART);
     AFTEREND = INDEX(LINEEND,'''');
     TITLE = SUBSTR(LINEEND,1,AFTEREND-1);
     CALL SYMPUT("TITLE&I",TITLE);
     RUN;
     %IF &SYSERR > 0 %THEN %DO;
     /* code here to handle missing graph program situation */
     %END;
%END;
%MEND  GETTTLS;

%GETTTLS /* get the graph title text for use in the selection menu */
RUN;

%MACRO SLCTLNS; /* 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;
%MEND  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 the %WINDOW statement */
```

Remaining application code uses this window definition, but is not presented here.

Rough Illustration of Graph Menu Window (not actual screen print):