

Guidelines for Development of SAS/IntrNet® Applications

Thomas J. Winn, Jr.
State Auditor's Office, Austin, Texas

Abstract

Application developers often have different backgrounds, training, development expertise, and coding styles. All of these factors influence the programmer's approach to application development. For most information technology departments, the resources expended for software maintenance constitute a huge share of the software budget, and software maintainers spent most of their time trying to understand the underlying logic of the system being modified.¹ Therefore, in order to make our coding more consistent, and to make the code that we write easier to maintain, we should seek adherence to a comprehensive set of development standards. This paper will discuss the establishment of standards and guidelines for the development of SAS/IntrNet® applications at the State Auditor's Office.

The Case for Standards

There are very many ways to write programming code in order to accomplish a particular objective. Application developers are creative, and like other artists each brings his/her own style and personality to the medium in which he/she works. Moreover, most developers dislike working on code written by others because they inevitably find examples of code they would have written differently. Because we all are different, it is important that we adhere to a comprehensive set of development standards to provide more consistency in our approach to software development.

Coding standards help to homogenize code. Subtle inconsistencies can creep into source code written by different authors. When working with code written according to an accepted standard, developers feel right at home, rather than being tempted to re-work all the code they touch to more fully align it with their personal tastes. When they are properly defined, trained, and enforced, coding standards can significantly reduce the cost of software development. Standards can also make coding easier to maintain and can improve the readability of code across all projects in an organization. Ultimately, adherence to standards has a positive impact on the bottom line cost of development.

The consistency provided by standards yields less tangible benefits as well, though perhaps no less important. For example, code that follows standards typically has fewer bugs and the bugs that do exist are easier to find and fix. So, having standards can help improve the quality of the code as well. This does not

¹ Gerardo Canfora and Aniello Cimitile, "Software Maintenance," p. 4,
<http://www.compaid.com/caiInternet/ezine/maintenance-canfora.pdf>

guarantee that code that conforms to standards is bug-free, but the consistency can make bugs easier to spot during the coding phase. The sooner a bug is found, the cheaper it is to fix. It has been estimated that 60% to 80% of the cost of a software development project is expended for maintenance.² Furthermore, 50% to 90% of the time spent on maintenance is spent on trying to comprehend the program logic; one of the reasons this is so problematic is that most of the maintenance is actually being done by programmers other than the original developers.³

Application Development Standards at the State Auditor's Office

For all of the aforementioned reasons, the .NET programmers at the Texas State Auditor's Office (S.A.O.) recently adopted standards and guidelines for their application development efforts to follow.⁴ Recognizing that the need for standardization in programming also is applicable to SAS®, but that SAS is a different development environment than .NET, the SAS/IntrNet developers at the S.A.O. decided to adopt their own development standards.⁵ This paper describes some of the elements of those standards and guidelines.

At the S.A.O., there are four SAS/IntrNet developers, and we have constituted ourselves into a SAS/IntrNet Development Team – this team is not a component of the agency's formal organizational structure, but is more like an informal advisory group. The team members prepared the SAS/IntrNet development standards together, and they also confer with one another frequently about the SAS/IntrNet projects that each of them is working on. We have found that this informal professional partnership provides constructive support to the development work that each member is responsible for, as well as promotes greater consistency in our coding practices. Moreover, since all of our intranet applications are very similar in their “look and feel”, it makes it easier for users to use them. The users don't have to accommodate themselves to a completely different operational framework as they move between our applications.

SAS/IntrNet Software

SAS/IntrNet software includes two CGI programs: Application Dispatcher (which facilitates the execution of SAS programs on request), and htmSQL (which allows the programmer to embed SQL queries inside HTML pages without using SAS DATA step code or SAS PROC steps). Therefore, applications can be developed using either Compute Services (Application Dispatcher), or Data Services (htmSQL). Here is a conceptual diagram which describes the

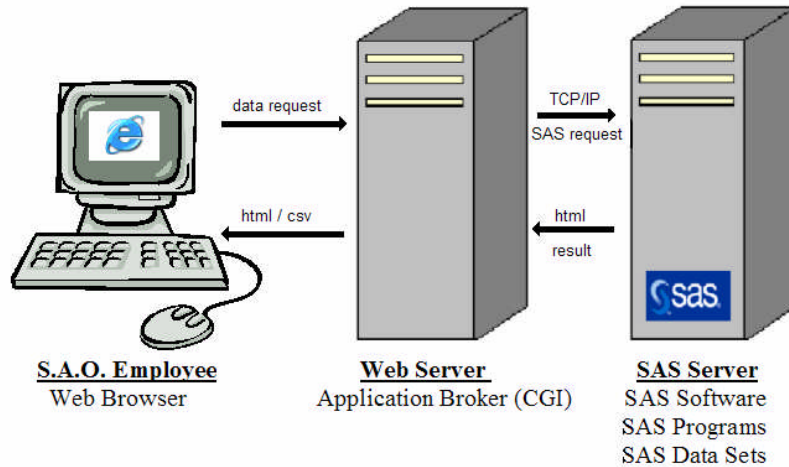
² Ibid.

³ Ibid., pp. 4-5.

⁴ Texas State Auditor's Office, “Information Systems Support Team Standards and Guidelines for Application Development”, March 3, 2006.

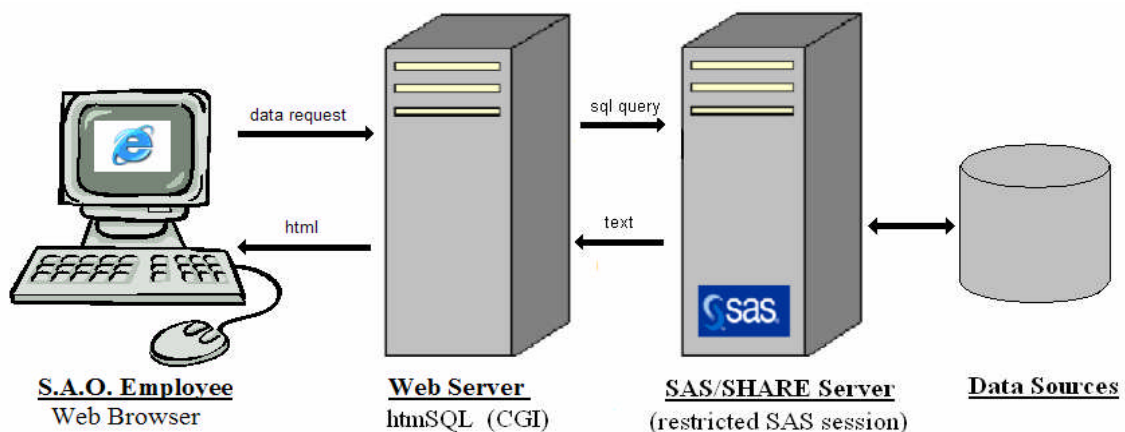
⁵ Texas State Auditor's Office, “Standards and Guidelines for Development of SAS/IntrNet Applications,” April 20, 2006.

interaction of operational components in SAS/IntrNet applications at the S.A.O. that use Compute Services with the Application Dispatcher:⁶



Application Dispatcher is composed of two pieces: (1) the Application Broker, which is a Common Gateway Interface (CGI) program that resides on the web server, and communicates between the browser and the SAS Application Server, and (2) the SAS/IntrNet compute server, which is a SAS program on the SAS Server.

If the organization has SAS/SHARE installed on the SAS Server, it also is possible for SAS programmers to develop SAS/IntrNet applications that make use of Data Services through execution of the htmSQL CGI program on the Web server. The conceptual diagram for Data Services is similar to the above diagram, except that the Web Server uses the htmSQL CGI program instead of the Application Broker, and the Web Server sends a query to the SAS/SHARE Server, which does not function as a full-blown SAS session, and from which the results are returned.⁷



⁶ SAS Institute Inc., SAS WebTools: Static and Dynamic Solutions Using SAS/IntrNet Software Course Notes, (Cary, NC: SAS Institute Inc., 2001), pp. 200-205.

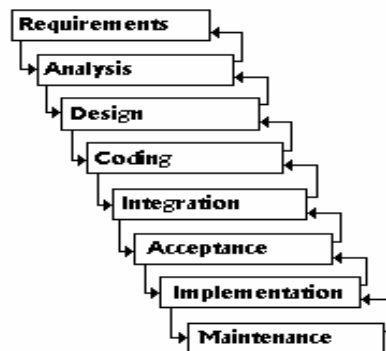
⁷ SAS Institute Inc., op. cit., pp. 196-198.

In general, SAS/IntrNet applications would be used whenever information is to be distributed via the Web (usually the SAO intranet), and whenever the data files used by the application are stored on the SAS Server. At the S.A.O., this approach is particularly useful for very large SAS data sets that were extracted from statewide databases residing on other agencies' computers.

If a developer has decided to use SAS/IntrNet, then we may assume that the new system will be a Web, and not a Windows, application. Before undertaking this type of development, the developer must be thoroughly familiar with SAS programming (including ODS), has a basic understanding of HTML, and also should have taken at least one of the "SAS Web Tools" training classes offered by SAS Institute, or equivalent training from another provider. As soon as possible, the developer should contact the organization's Internet/Intranet administrator (or its webmaster), to determine where the project files will be created on the Web Server(s), and also to review the organization's web page standards. If the developer does not have an HTML editing tool (such as Macromedia Dreamweaver®) already installed on his/her computer, then he/she should arrange for that installation.

System Development Life Cycle Methodology

Wherever feasible, developers of application systems are encouraged to use a System Development Life Cycle (SDLC) Methodology.⁸ Although there exist numerous versions of SDLC, the typical methodology includes multiple stages in which the output of each stage becomes the input for the next stage, as in the following diagram:



Typical System Development Life Cycle Methodology

The steps are:

- (1) Problem Definition – Identify the users' requirements – this is a statement of needs, from the users' perspective,

⁸ Systems Development Life Cycle, http://en.wikipedia.org/wiki/System_Development_Life_Cycle, and <http://www.computerworld.com/developmenttopics/development/story/0,10801,71151,00.html>

- (2) Preliminary Analysis and Planning – Refine the common understanding (between users and programmers) concerning what the program or system is supposed to do – these are statements containing the functional specifications, the feasibility of the project, the proposed timeline, the resources needed, and a consideration of alternative solutions,
- (3) Design – Determine how the program or system will get it done -- create detailed specifications for each process, the development work plan, and a protocol for validation,
- (4) Coding and Testing of the components – programming activities,
- (5) Integration with other systems, and more Testing – check for errors, bugs, and interoperability – *moreover, it is suggested that before user acceptance testing, the almost-completed test application be subjected to peer review by other members of the SAS/IntrNet Development Team,*
- (6) Obtain User Acceptance – this involves more Testing before user signoff for the deliverables,
- (7) Implementation, documentation, and user training – deployment of the system,
- (8) Post-Implementation Analysis, Maintenance, and Change Management.

These steps may be compressed to include only those activities that would be appropriate for a particular project. However many steps there are, each of them would include testing, to ensure that the desired outcomes were achieved. And each step also would seek an agreement of all of the parties involved regarding the outcomes. The essential walkthroughs will help to clarify understandings. Furthermore, since SAS/IntrNet software provides a rapid development environment, it is possible for a SAS/IntrNet developer to use prototyping so as to interactively engage a few key users in the design process, while some of the preliminary coding is taking place. And this expedites user acceptance later on.

Making Choices About Functionality

As mentioned previously, two of the programming approaches available in SAS/IntrNet are Application Broker (which makes extensive use of the SAS DATA step, SAS procedures, and ODS) and htmSQL (which is based on the idea of embedding SQL queries in HTML, and formatting the results according to the HTML code instead of using ODS). Either of those approaches can be enhanced by the use of JavaScript. Alternatively, if SAS Integration Technologies has been licensed by the organization, then developers could use Java Server Pages to build a Web service involving SAS, or to access SAS using the Microsoft .NET framework.

Deciding between the Application Broker and htmSQL programming approaches will generally be determined by the desired functionality of the application. For simple user selections, either approach probably could be used. If the developer has strong skills with programming DATA steps, procedures, and ODS, then

Application Broker might be the preferred choice; but if the developer knows all about SQL, then htmSQL could be best. If the menu choices offered to users will be fairly simple, and if the users will not be permitted to specify which variables will be populated in the result set, then Application Broker offers a straightforward programming approach. On the other hand, if there is a need to dynamically populate the HTML form fields from the front-end, which will make for more complicated text boxes for the user selections, then htmSQL would be used instead.

How will user-supplied parameters be validated? With SAS in the dispatcher program? With JavaScript on the client side, before submitting the dispatcher request? Or with a Java applet on the client side?

What kinds of output will the application generate? Detailed or summary data? Listings, tabular reports, or graphics? Those choices should affect how the data are organized, as well as how the code is written.

Will optional, additional information be made available regarding the data, the application, or the interpretation of the results, by clicking a button on certain screens?

For longer-running queries, what message(s) will be sent to the user, to reassure him/her that the request is being processed?

To a large extent, the application's response time will be determined by the size of the data files. Will the entire result set be returned to the user each time a query is executed? Or would only some of the observations be returned to the user, pending a user decision that the results appear to meet his/her expectations?

What type of message will be returned to the user if no observations were found which satisfy the user's specified selection criteria?

Will links be provided, to facilitate navigation to related web pages or applications? Although it may not be a requirement at all installations, it is preferred that HTML documents used in SAS/IntrNet applications contain links to other pages which are its hierarchical precedents [that is, going back to the default page for the application, and also just beyond that to the root page which brought the user to the application]. Those navigational links should be placed just below the style sheet banner, and just before the H1 main heading in the Body section of the HTML document. Each navigational link is identified by a tag, and uses a relative URL to specify the location. Alternatively, the developer could use prominently-displayed buttons as navigational aids. Resizable "child windows" should not include navigational aids, other than a "Close Window"-button.

Standards and Guidelines for SAS Code Used in Dispatcher Programs⁹

Names

Use meaningful, unambiguous names for libraries, programs, files, data sets, variables, formats, macros, and macro variables. Well-chosen names furnish important information, which may help the reader to grasp the meaning and intent of the code in which they are used. Generic names are okay, if circumstances make them necessary (for example, when you have a lot of very similar things to be identified), but it is better to use short, descriptive names, wherever possible. Names like A, B, C, X1, X2, X3, SPARKY, MUFFY, FLOPSY, or TODDLES are meaningless, and provide the reader with no information.

In Version 6 and earlier releases of the SAS System, the maximum length of SAS names was 8 characters. With Versions 8 and 9, most SAS names may be up to 32 characters in length. It's nice to know that we now have so much flexibility in naming. Nevertheless, our SAS programmers still find that short names are easier to manage. Try to find the right balance between brevity and meaningfulness. And be consistent. If you have several programs which use the same data items, then use the same names in each program.

Here are some standards pertaining to naming:

- In naming, avoid cuteness, single-letter names, and names that too closely resemble one another.
- Names should be unique, short, and descriptive – in that order of importance.
- If longer names are needed, underscores may be used to separate words, in order to enhance readability.

Examples: DEBT_TO_ASSETS, EMPL_GROWTH_RATE, OP_INCOME_TO_ASSETS, INVENTORY_TO_SALES, LOG_OF_SALES, INVENTORY_TURNOVER, RETURN_ON_EQUITY.

- If a user-defined format applies to only one variable, then name the format with a readily-recognizable form of the variable-name plus the suffix FMT . For example,

```
PROC FORMAT;  
  VALUE $SEXFMT 'F' = 'FEMALE'  
               'M' = 'MALE';
```

- Programs which are sequential should share the same name-prefix, but also have a numerical suffix which indicates their place in the sequence.
- Names of data files, which are shared by a group of individuals, should adhere to a naming standard which suits the needs of the group and the

⁹ Thomas J. Winn Jr., "Guidelines for Coding of SAS Programs", Proceedings of SCSUG 2003 (sponsored by the South-Central SAS Users' Group, and held on October 26-28, 2003), pp. 320-325; and Proceedings of the 29th Annual Conference of the SAS Users Group International (2004), Paper 258-29.

context of the data. For example, name components might include distinct identifiers for application, module, modifier, source, date-time, user-ID, sequence-number, and type-extension. Develop a standard that works, and use it consistently.

Readability & Appearance

Write code that is readable. It will be easier to understand and to modify.

- Programs should look neat and orderly,
- The visual appearance of a program listing should mirror its logical structure,
- Use white space liberally.

Separate blocks of code, using indents and white space.

- ◆ Insert a blank line between SAS program steps; that is, before each DATA or PROC step.
- ◆ Be consistent with your indentation increments.
- ◆ Indent all statements in a logical grouping by the same amount.
- ◆ Left-justify all OPTIONS, DATA, PROC, and RUN statements. Indent all of the statements within a DATA or PROC step.
- ◆ Indent conditional blocks and DO groups, and do it consistently. The logic will be easier to follow.
- ◆ Align each END statement with its corresponding DO statement. This will make it easier to verify that they match.
- ◆ Remember to preface major blocks of code with explanatory comments.
- ◆ Consider inserting PAGE statements to force the SAS Log to begin tracing the execution of new modules on a new page.

Furthermore,

- Don't place more than one programming statement on a single line.
- Make sure that titles and footnotes used in printed reports accurately reflect the information being presented.
- Make sure to use date formats when reporting SAS dates.

Here are a few lines of code from a SAS job:

```
PROC SUMMARY DATA=VND RPMTS NWAY MISSING; WHERE AGENCY=324 AND  
( '01SEP2002'D<=EFF_DATE<='31AUG2003'D); CLASS OBJECT VEN_NO; ID  
VEN_NAME; VAR AMOUNT; OUTPUT OUT=VNDRSMRY.AGY324(DROP=_TYPE_ _FREQ_)  
SUM=; RUN; PROC PRINT DATA=VNDRSMRY.AGY324 N UNIFORM; VAR OBJECT  
VEN_NO VEN_NAME AMOUNT; SUM AMOUNT; FORMAT AMOUNT DOLLAR15.2; RUN;
```

The following lines would be much easier to follow:

```
*** CREATE A TAPE FILE CONTAINING SUMMARIZED PAYMENTS *****;  
PROC SUMMARY DATA=VND RPMTS NWAY MISSING;
```

```

WHERE AGENCY=324 AND ('01SEP2002'D <= EFF_DATE <='31AUG2003'D);
CLASS OBJECT VEN_NO;
ID VEN_NAME;
VAR AMOUNT;
OUTPUT OUT=VNDRSMRY.AGY324(DROP=_TYPE_ _FREQ_) SUM=;
RUN;

*** PRODUCE THE VENDOR PAYMENTS SUMMARY REPORT *****;
PROC PRINT DATA=VNDRSMRY.AGY324 N UNIFORM;
  VAR OBJECT VEN_NO VEN_NAME AMOUNT;
  SUM AMOUNT;
  FORMAT AMOUNT DOLLAR15.2;
  TITLE1 'SUMMARY OF SELECTED VENDOR PAYMENTS FOR AGENCY #324';
  TITLE2 'DURING FY2003, ACCORDING TO OBJECT-CODE';
RUN;

```

- ◆ Insert parentheses in meaningful places in order to clarify the sequence in which mathematical or logical operations are performed,
- ◆ Break really complicated statements into a number of simpler statements,
- ◆ Don't be too clever,

The SAS System certainly makes computing easy for the programmer. It provides data set names, as well as default types, widths, and formats for variables, even though the programmer may have neglected to code these items explicitly. Unless the programmer specifies otherwise, the SAS System assumes that all of the observations for all of the variables should be included in the output data set. And if the programmer fails to identify a particular SAS data set when invoking a SAS procedure, then the SAS System assumes that the programmer intended that the PROC should be executed using the most-recently-created data set within the job. Each SAS procedure has default options. This is all very convenient, so long as the actual outcome is what was intended. The advice is, know your defaults, but don't overdo invoking them implicitly. There needs to be a balance between programmer convenience and program completeness and clarity. The fundamental idea is to be intentional about telling the SAS System what you want, and not just to rely passively on SAS defaults. A reader of your code should be able to fully and correctly comprehend what you wrote. Don't be too clever. The programmer should strive to make the program easy to understand, and care also needs to be taken to ensure that the wrong data set isn't used, by accident.

Internal Program Documentation

Use numerous explanatory comments within the program,

- ◆ Use comments to provide a program header, a prologue which describes what the program was intended to accomplish, its author, and the circumstances of its having been written (what, why, for whom, by whom, and when); the prologue also should include a change control log, which describes subsequent modifications made to the program (what, why, for whom, by whom, and when),

- Here is a skeletal example of a program header. It would be placed very near to the beginning of the program:

Here is an example which illustrates the two different types of comments which may be used in a SAS job
(* comment; and /* comment */):

10

```

***          OF COMPUTER PROGRAMMING, VOLUME 3, SORTING AND
***,
***          SEARCHING", PUBLISHED BY ADDISON-WESLEY IN 1973. - - -
***;
%MACRO SOUNDZ;
  DATA NONMATCS;
    SET NONMATCS;
    MATCSND = 0; /* COUNTER FOR ITEMS THAT MAY SOUND ALIKE */
    %DO I=1 %TO 9;
      IF SOUNDEX(NAME&I._&CH1) NE ' ' AND
      ( INDEX(TRIM(SOUNDEX(NAME1_&CH2)),TRIM(SOUNDEX(NAME&I._&CH1)))>0
        OR INDEX(TRIM(SOUNDEX(NAME2_&CH2)),TRIM(SOUNDEX(NAME&I._&CH1)))>0
        OR INDEX(TRIM(SOUNDEX(NAME3_&CH2)),TRIM(SOUNDEX(NAME&I._&CH1)))>0
        OR INDEX(TRIM(SOUNDEX(NAME4_&CH2)),TRIM(SOUNDEX(NAME&I._&CH1)))>0
        OR INDEX(TRIM(SOUNDEX(NAME5_&CH2)),TRIM(SOUNDEX(NAME&I._&CH1)))>0
        OR INDEX(TRIM(SOUNDEX(NAME6_&CH2)),TRIM(SOUNDEX(NAME&I._&CH1)))>0
        OR INDEX(TRIM(SOUNDEX(NAME7_&CH2)),TRIM(SOUNDEX(NAME&I._&CH1)))>0
        OR INDEX(TRIM(SOUNDEX(NAME8_&CH2)),TRIM(SOUNDEX(NAME&I._&CH1)))>0
        OR
      INDEX(TRIM(SOUNDEX(NAME9_&CH2)),TRIM(SOUNDEX(NAME&I._&CH1)))>0)
      THEN MATCSND = MATCSND + 1;
    %END;
  RUN;
%MEND SOUNDZ;
%SOUNDZ

```

And here is yet another example which illustrates the two comment-types:

```

*** WRITE SELECTED PAYROLL INFORMATION TO A FLATFILE ON DASD
*****;
DATA _NULL_;
  SET COMB02;
  FILE PAYDATA;
  PUT @1    CMPNUM          3.0 /* AGENCY NUMBER */
      @4    RPTNAME        $67. /* EMPLOYEE NAME */
      @71   EMPLOYEE       SSN11. /* EMPLOYEE S.S.N. */
      @82   CHKDATE        YYMMDD10. /* PAYMENT DATE */
      @92   CKSEQ          3.0 /* CHECK SEQUENCE NUMBER */
      @95   LABRSEQ        3.0 /* COMPONENT SEQUENCE NUMBER */
      @98   CHECKNO        11.0 /* PAYMENT NUMBER */
      @109  PEREND         YYMMDD10. /* PAYROLL PERIOD ENDING DATE */
      @119  PCA            $5. /* PROGRAM COST ACCOUNT */
      @124  OBJCD          $4. /* COMPTROLLER OBJECT CODE */
      @128  DOCNUM         $8. /* DOCUMENT NUMBER */
      @136  FUND           $4. /* FUND NUMBER */
      @140  FY             $2. /* FISCAL YEAR */
      @142  AY             $2. /* APPROPRIATION YEAR */
      @144  GROSS          10.2 /* GROSS AMOUNT OF PAYMENT */
      @154  ST_RET         10.2 /* STATE-PAID RETIREMENT AMOUNT */
      @164  OASDITAX       10.2 /* STATE-PAID OASDI TAX */
      @174  HICOTAX        10.2 /* STATE-PAID HI MEDICARE TAX */
      @184  ST_INS         10.2 /* STATE-PAID HEALTH INSURANCE */
      @194  FLAGB          $RETFMT. /* RETIREMENT PLAN */ ;

```

Reusability

Write the code that you use repeatedly as a macro, and then, instead of repeating your code, just invoke the macro. But beware! Since most of the operations of the SAS macro facility are carried out in the background, sometimes debugging them can be fairly mysterious. It is especially important for macros to be well-documented.

- Write code that can be re-used, with different parameters. Keyword parameters are preferable to positional parameters, because they are less likely to be specified incorrectly.
- Write the code you use repeatedly as a macro, and then, instead of repeating your code, invoke the macro.
- Avoid using global macro variables.
- If a macro is used by more than one program, put it into an AUTOCALL macro library.

Efficiency

One method of improving the human efficiency of the programs that we write is to use a linearized, modular style of programming. Programs written in this manner generally require less programmer time for debugging, updating, or modifying.

- ◆ Avoid jumping to statement labels by GO TO, or LINK statements and RETURN statements,
- ◆ If possible, replace logic which jumps between subroutines with DO ...END and IF ... THEN ... ELSE ...-logic,
- ◆ End every DATA and PROC (except PROC SQL) step with a RUN statement,
- ◆ End every PROC SQL step with a QUIT statement.

Here is an example of a portion of a DATA-step which uses GO TO statements and subroutine labels in such a way as to cause the SAS System to jump to a new location, and to continue execution from that point onward:

```

LENGTH BEGINP 4  STOPP 4  SORTKEY $19;
IF BEGPER < 1000 THEN GO TO L350;
  ELSE GO TO L355;
RETURN;
L350: STARTQTR = BEGPER;
      STARTMO = 1;
      BYY = INT(BEGPER/10);
      BQ = BEGPER - (10*BYY);
      IF BQ > 0 THEN BMM = (3*BQ) - 2;
        ELSE IF BQ = 0 THEN BMM = 1;
      GO TO L360;
L355: BYY = INT(BEGPER/100);
      BMM = BEGPER - 100*BYY;
      STARTQTR = INT(((BMM + 2) / 3) + (BYY*10));
      IF          BMM IN (1,4,7,10) THEN STARTMO = 1;
        ELSE IF BMM IN (2,5,8,11) THEN STARTMO = 2;
        ELSE IF BMM IN (3,6,9,12) THEN STARTMO = 3;

```

```

        GO TO L360;
L360: BEGINP = COMPRESS(STARTQTR || STARTMO);
      IF ENDPER < 1000 THEN GO TO L370;
        ELSE GO TO L375;
      RETURN;
L370: ENDQTR = ENDPER;
      ENDMO = 4;
      EYY = INT(ENDPER / 10);
      EQ = ENDPER - (10*EYY);
      IF EQ > 0 THEN EMM = 3*EQ;
        ELSE IF EQ = 0 THEN EMM = 12;
      GO TO L380;
L375: EYY = INT(ENDPER / 100);
      EMM = ENDPER - (100*EYY);
      ENDQTR = INT(((EMM + 2)/3) + (EYY*10));
      IF      EMM IN (1,4,7,10) THEN ENDMO = 1;
        ELSE IF EMM IN (2,5,8,11) THEN ENDMO = 2;
        ELSE IF EMM IN (3,6,9,12) THEN ENDMO = 3;
      GO TO L380;
L380: STOPP = COMPRESS(ENDQTR || ENDMO);
      SORTKEY = COMPRESS(TPNUM || BEGINP || STOPP);
      KEEP TPNUM      BEGPER      ENDPER      BEGINP      STOPP
          SORTKEY      TOTHR      TRESULTS;

```

The preceding example uses a legitimate programming style, but the approach can lead to convoluted statements which may be difficult for a reader to follow. Actually, the preceding example isn't too arcane – it is modular, and fairly linear. However, it could have been made a lot more confusing by making additional separately-labeled subroutines out of the conclusions of each conditional statement, and then by shuffling all of the subroutines in the DATA- step. But you get the idea.

Now, here is the same program component, except that the execution is controlled by DO-END and IF-THEN-ELSE programming statements, which most SAS programmers find are easier to read and to understand:

```

LENGTH BEGINP 4  STOPP 4  SORTKEY $19;
IF BEGPER < 1000 THEN DO;
  STARTQTR = BEGPER;
  STARTMO = 1;
  BYY = INT(BEGPER/10);
  BQ = BEGPER - (10*BYY);
  IF BQ > 0 THEN BMM = (3*BQ) - 2;
    ELSE IF BQ = 0 THEN BMM = 1;
END;
ELSE DO;
  BYY = INT(BEGPER/100);
  BMM = BEGPER - 100*BYY;
  STARTQTR = INT(((BMM + 2) / 3) + (BYY*10));
  IF      BMM IN (1,4,7,10) THEN STARTMO = 1;
    ELSE IF BMM IN (2,5,8,11) THEN STARTMO = 2;
    ELSE IF BMM IN (3,6,9,12) THEN STARTMO = 3;
  END;
IF ENDPER < 1000 THEN DO;

```

```

        ENDQTR = ENDPER;
        ENDMO = 4;
        EYY = INT(ENDPER / 10);
        EQ = ENDPER - (10*EYY);
        IF EQ > 0 THEN EMM = 3*EQ;
           ELSE IF EQ = 0 THEN EMM = 12;
    END;
    ELSE DO;
        EYY = INT(ENDPER / 100);
        EMM = ENDPER - (100*EYY);
        ENDQTR = INT(((EMM + 2)/3) + (EYY*10));
        IF          EMM IN (1,4,7,10) THEN ENDMO = 1;
           ELSE IF  EMM IN (2,5,8,11) THEN ENDMO = 2;
           ELSE IF  EMM IN (3,6,9,12) THEN ENDMO = 3;
    END;
    BEGINP = COMPRESS(STARTQTR || STARTMO);
    STOPP = COMPRESS(ENDQTR || ENDMO);
    SORTKEY = COMPRESS(TPNUM || BEGINP || STOPP);
    KEEP TPNUM  BEGPER  ENDPER  BEGINP  STOPP
        SORTKEY  TOTHR5  TRESULTS;

```

Group the executable statements. Place most of the non-executable in a DATA step before all of the executable statements. In particular, place variable attribute and other declarative statements near to the top of the DATA step, and ahead of the executable statements. There is a difference between declarative programming statements which deal with descriptor information (like ATTRIB, LENGTH, INFORMAT, FORMAT) or other declarative statements (like ARRAY, RENAME, RETAIN), and which take effect when SAS compiles the program, and programming statements which actually affect DATA step execution (like INPUT, INFILE, SET, MERGE, DO, IF, IF-THEN-ELSE, PUT, RETURN, and assignment statements). It is a good idea to group most of the declarative statements together at the beginning of the DATA step, to keep them from cluttering up the logic portion of the program. However, it also seems logical to place DROP or KEEP statements after the executable statements, instead of before them. One advantage of following this convention of putting like things together is that the reader could get a good idea which variables are created and their characteristics just by scanning the first few lines in a DATA step. An additional advantage of putting like things together is that it is easier to check for completeness and to find certain types of errors.

Here are some more standards that pertain to efficiency:

- In a DATA step, place most of the non-executable statements before the executable statements – exceptions include the DROP or KEEP statements, which may be placed after the executable statements.
- Define INPUT and PUT variables one per line, using @ pointer control.
- Screen data for unusual circumstances.

Errors often can be avoided by checking the data for invalid values.

Unfortunately, erroneous or unforeseen data abound throughout the world, and your SAS program will find some of them. Experience can be your best teacher.

Pay attention to the unexpected circumstances that aren't supposed to happen, but do, in the particular data files that you use. It is a good idea for you to do some preliminary data exploring with PROC FREQ, to become better acquainted with your data, before you put that new, special report into production. If the subject-matter experts tell you that only n possible conditions can be applicable in some given situation, you should be skeptical enough to expect more than that, and code accordingly.

The most important thing most SAS users can do to improve the computer efficiency of their programs is to decrease the number of input/output operations. Strive to minimize the number of passes through the data, as well as to minimize the number of DATA steps. If multiple data sets are to be created from the master data set, then create them all with one DATA step. The use of intermediate DATA steps to subset the data, to transform the data, or to perform calculations should be curtailed. Instead of using a subsetting-IF statement in a DATA step as a preliminary to running some SAS procedure, eliminate the need for a DATA step entirely by performing the subsetting in the PROC step, through the use of a WHERE statement. Wherever possible, take advantage of procedure output data sets as inputs to other PROCs, to avoid having to read the master data set again. Read and store only the fields that you need. And sort the data only when it is absolutely necessary.

In summary, to reduce the number of times the data are read:

- Minimize the number of passes through the data,
- Minimize the number of DATA steps,
- Read and store only the data that are needed,
- Sort the data only when it is absolutely necessary.

Here are a few more efficiency-related guidelines:

- When you read in an external file, use pointer controls, informats, or column specifications in the INPUT statement, to read only those fields you actually need.
- Store only the variables you need by using DROP or KEEP statements, DROP= or KEEP= options (eliminate variables from the output data set which are needed only during DATA step execution, and not afterward).
- When only one condition can be true for a given observation, use IF ... THEN ...ELSE ... statements (or a SELECT group), instead of a series of IF ... THEN ... statements without ELSE statements (In a sequence of IF-THEN statements without the ELSE, the SAS System will check each condition for every observation).
- When using a series of IF ... THEN ... ELSE ... statements, list the conditions in descending order of probability. This will save CPU time.
- Use the LENGTH statement to reduce the storage space for variables in SAS data sets.

- Minimize workspace usage by using the DELETE statement in a PROC DATASETS step, to eliminate temporary data sets that are no longer needed by the program.
- Use the IN operator instead of a series of multiple logical OR operators.

Naming Conventions and Storage Locations

SAS Data Files

SAS data files, including native SAS data sets (formatted by SAS) and interface SAS data sets (formatted by non-SAS software, and accessed through an engine), are stored in data libraries, which are folders located on the SAS Server, as designated by the SAS Server Administrator. SAS/IntrNet developers may place data only in the data libraries that are authorized for their use. Each data file shall have a unique name, which may not exceed 32 characters in length, should begin with a letter, and subsequent characters may be letters, numeric digits, or underscores. Names may not contain any blanks or special characters (other than underscores). The names are selected at the discretion of the SAS/IntrNet developers. These should be meaningful names, that follow some systematic pattern, and that can readily be recalled by the SAS/IntrNet developer. The names should be simple but descriptive.

The names of the variables (columns) in the SAS data files must follow the same rules as for the names of SAS data files (must be unique, no more than 32 characters, beginning with letter, containing no special characters except underscores). If more than one SAS data file in the data library contains columns whose values pertain to the same kind of information and have the same attributes, then the corresponding columns should be given the same column name in each data SAS data file. SAS/IntrNet developers should discuss data needs with the SAS Server Administrator prior to development of applications to discuss table and column naming conventions and to ensure consistency with existing tables.

SAS Dispatcher Programs

SAS dispatcher programs that are invoked by the Application Broker are stored as members of SAS program libraries, which are folders located on the SAS Server, as designated by the SAS Server Administrator. SAS/IntrNet developers may place programs only in the program libraries that are authorized for their use. Each dispatcher program shall have a unique name, which may not exceed 32 characters in length, should begin with a letter, and subsequent characters may be letters, numeric digits, or underscores. Program names may not contain any blanks or special characters (other than underscores). The program names are selected at the discretion of the SAS/IntrNet developers. These should be meaningful names, that follow some systematic pattern, and

that can readily be recalled by the SAS/IntrNet developer. SAS dispatcher program names use the **.sas** extension. The names should be simple but descriptive.

HTML Documents

Web pages are HTML documents that consist of text and HTML tags. A Web browser interprets the tags in an HTML document and displays the document as a web page. HTML documents are used to welcome users, to direct them via links to other HTML documents that pertain to their areas of interest, and to generate forms which gather information that is entered by users. Currently, the standard specification for HTML is HTML 4 (either HTML 4.0 or HTML 4.01 may be used at the SAO). Element names are written using uppercase letters, and attribute names are written using lowercase letters. Most HTML tags have an opening tag and a closing tag, although some tags have only an opening tag. On the World Wide Web, HTML is evolving into an XML-based markup language (XHTML2). HTML documents are stored on the specified Web Server as members of folders, located on the production Web Server (or located in folders on the test Web Server), that are designated for the particular SAS/IntrNet application. At the SAO, HTML document names must begin with a letter; subsequent characters may be letters, numeric digits, hyphens, or underscores; they must contain no embedded blanks; and they must use the **.html** extension. The first point of entry (home page) for each application is named **default.html**. HTML document names are selected at the discretion of the SAS/IntrNet developers. These should be meaningful names, that follow some systematic pattern, and that can readily be recalled by the SAS/IntrNet developer. The names should be simple but descriptive. It is suggested that HTML documents be named in such a way as to distinguish between HTML forms, "additional information" windows, and other types of HTML documents.

htmSQL Input Files

An htmSQL input file contains HTML with SQL queries and statements embedded in it. It allows a SAS/IntrNet developer to embed data from SAS data sets and external DBMSs into a Web page. The embedded Web references may be to either static or to dynamic content. In an htmSQL input file, the SQL statements (which end with a semicolon) appear in a section between the {sql} opening tag and the {/sql} closing tag. htmSQL input files follow the same naming conventions as SAS dispatcher programs, except that they use the extension **.hsql**. At the SAO, htmSQL input files are stored on the specified Web Server as members of a subfolder exclusively for htmSQL files, *within the designated folder on the Web Server for the particular SAS/IntrNet application*.

JavaScript

JavaScript is a client-based Web scripting language that is used in conjunction with HTML to provide interactivity to Web pages. JavaScript can be embedded

within HTML documents between the <script> opening tag and the </script> closing tag, or JavaScript may be stored in external files which can be linked to the HTML document through the use of the src attribute in the <script> tag. At the SAO, external JavaScript files follow the same naming conventions as SAS programs, except that they use the extension .js; and external JavaScript files are stored on the specified Web Server as members of a subfolder exclusively for JavaScript files, *within the designated folder on the Web Server for the particular SAS/IntrNet application.*

Handling Errors¹⁰

All code contains errors of one kind or another and how you deal with errors may be the most important part of a well-designed application. There are two types of errors: those you can prevent, which are called development errors, and those you can't prevent but may be able to identify, which are called run-time errors.

Development errors are either syntax errors or logic errors. Syntax errors occur from typographical errors, missing punctuation, or improper use of a language element. SAS Logs do a good job of identifying syntax, punctuation and some typographical errors (notes and warnings will tell you about misspelled keywords and inconsistent data types). If you suspect that you are having a problem with macro content, don't forget to use the macro debugging options. Logic errors are more commonly referred to as bugs or undocumented features. These errors occur when code executes without causing an error, but does not produce the results intended. SAS cannot identify errors in logic. If you tell SAS to do the wrong thing in a syntactically correct way, SAS will assume that you know what you are doing and it will continue executing. You eliminate development errors by "debugging" your code.

During development and testing of the application, a very useful technique for debugging SAS programs invoked through the Application Dispatcher is to append the debug flag and associated parameter value "&_debug=131" to the URL. This returns all values passed to the SAS Server, the SAS Log, and the total elapsed time.

Whenever a run-time error occurs, the SAS System will display an error message and terminate the application. While this behavior may be acceptable when you are writing and debugging code, it is never acceptable when users are running your application.

One way of avoiding run-time errors is to anticipate problems arising from incorrect or missing input parameters provided by the user. User inputs are beyond the programmer's immediate control, and they will prevent the code from

¹⁰ Texas State Auditor's Office, "Information Systems Support Team Standards and Guidelines for Application Development", op. cit.

executing correctly at run time. Therefore, SAS/IntrNet developers are strongly encouraged to include routines for validating user responses. This can be handled from the client side, through JavaScript or a Java applet in the HTML document on the Web Server, or on the SAS Server through routines which check the input parameters at the beginning of the dispatcher program. Erroneous user inputs should generate an informative message sent to the Web Browser. This will give the user a chance to fix the error directly. Planning and designing such error handling routines is an integral part of SAS/IntrNet application development.

Utilizing Active Directory Services

If the organization uses Windows Active Directory services, then SAS/IntrNet developers have additional tools to assist in authenticating users and assessing their privileges when developing intranet applications. At the S.A.O., Active Directory Services are utilized by all of the agency's computers, and this allows the developer to more easily authenticate users (and can be used in lieu of login screens). Internally, Active Directory may reduce the need for anonymous logins and other forms of user authentication. In conjunction with Active Directory, it is possible to capture the LAN IDs of the users of your application. Please contact the web administrator before implementing Active Directory, since some settings may need to be changed on the web server and/or individual workstations for these services to work effectively.

Testing Your Application¹¹

Thorough application testing is an integral part of the SDLC process. Effective testing is used for quality assurance -- testing the accuracy, completeness, reliability, functionality, quality and performance of the entire application. Testing also helps ensure that applications include the expected functionality, security and internal control features required to function effectively. Testing can also be used to ensure the interoperability of applications and other network components. The developer and the user(s)/tester(s) should reach an agreement during the initial planning stage of the project on the level of testing to be performed. When feasible, a test plan should be developed for acceptance testing.

Once the developer has completed his initial testing, the application should be deployed to the test server for further testing. The web administrator can assist in the deployment process (deployment is discussed in the following section). The test server should be available to all developers, and users/testers can be granted access to the appropriate test area once the application is deployed. At the S.A.O., the test server simulates the production environment. Ensure that all files required to run the application have been identified for final deployment.

¹¹Ibid.

Extensive user testing is imperative to an application's success since it determines whether the application meets defined acceptance criteria. During testing, users often identify program defects or weaknesses. These errors must be communicated to the developer so that corrective action can be taken as soon as practical. Follow-up testing of application changes and enhancements should be performed and final acceptance (formal or otherwise) should be obtained before the application is deployed to the production server.

Different applications require different levels of testing, depending on their complexity, intended use, and the types of data each will be used to process. The following tests should be considered and applied as appropriate.

- **Acceptance Testing** - End users perform acceptance tests to assess the overall functionality and interoperability of an application.
- **End-to-End Testing** - End users and system technicians perform end-to-end tests to assess the interoperability of an application and other system components such as databases, hardware, software, or communication devices.
- **Functional Testing** - End users perform functional tests to assess the operability of a program against predefined requirements. Functional tests include black box tests, which assess the operational functionality of a feature against predefined expectations, or white box tests, which assess the functionality of a feature's code.
- **Integration Testing** – End users and system technicians perform integration tests to assess the interfaces of integrated software components.
- **Parallel Testing** - End users perform parallel tests to compare the output of a new application against a similar (original) application.
- **Regression Testing** – End users retest applications to assess functionality after programmers make code changes to previously tested applications.
- **Stress Testing** – Technicians perform stress tests to assess the maximum limits of an application.
- **String Testing** – Developers perform string tests to assess the functionality of related code modules.
- **System Testing** – Technicians perform system tests to assess the functionality of an entire system.
- **Unit Testing** – Developers perform unit tests to assess the functionality of small modules of code.

Application Deployment

Once application development and testing are complete, it must be deployed to the appropriate server. Deployment is the process of packaging and distributing an application and related dependency files to the appropriate users. Initial deployment of web applications should be coordinated with the web

administrator. Please contact him/her in advance for assistance in building the appropriate deployment strategy for your application. He will need to create virtual directories, folders and establish security profiles for the application on the host web server before the application can be deployed. Once this is done, copy the folders of HTML documents and htmSQL input files from the test environment to the production environment.

Post-Mortems

Did the application achieve the expected results? Did it meet the user's needs? With the increased emphasis on technology, deadlines, hour budgets, usage statistics, etc. developers should consider performing application post-mortems with the application owners to see if they are satisfied with the system as delivered. Post-mortems provide an excellent opportunity to discuss successes and failures with the project from the request through the implementation phases. This insight can help developers better plan their approach for future development efforts.

Documentation

After a SAS/IntrNet application is moved to production, technical documentation concerning the application is kept in electronic form in an application-specific folder located on the SAS Server, and including at least the following items:

- (a) Identification – the name of the application; the name(s) of the person(s) responsible for development and maintenance of the application; the name(s) of the application owner(s), and the intended audience of the application.
- (b) Requirements – a statement of needs, from a user's perspective, that the application was intended to satisfy.
- (c) Analysis and Design – a description of the functional specifications for the application, the derivation of the data used by the application, the processes used by the application, and the outputs produced by the application. This also will include a file layout or PROC CONTENTS output for each data set used by the application, a Microsoft Visio Web Site Map diagram (which identifies the specific SAS Dispatcher programs that are called by the Application Dispatcher), and data flow diagrams for illustrating the structure, relationships, and processing flow of any SAS Dispatcher programs which contain multiple DATA and/or PROC steps, and macros.
- (d) Testing – a description of procedures used for testing the application.
- (e) Location of Code Backups – the location on the network where backup listings of the data extraction programs, SAS dispatcher programs, HTML forms, htmSQL input files, external JavaScript files, and any supplemental files that are used by the application are stored.
- (f) Information for Users – informational materials used for publicizing the existence and use of the application, or for user training.
- (g) Data Maintenance -- A step-by-step description of procedures to be followed for updating the data used by the application. The procedures will ensure that

SAS Logs of the data extraction programs are reviewed, and that the numbers of records and any control totals are captured, reviewed, and retained.

Optional additional items which could be included in the documentation folder are:

- a narrative description of the circumstances concerning the application's development – meetings or consultations held, contacts made, time spent, problems encountered, communications, observations, etc.
- selected information concerning input file(s) – location, name, record layout, data dictionary report (if available), and typical data values.

As noted in item (e) above, it is important that the data extraction programs, SAS dispatcher programs, HTML documents, htmSQL input files, external JavaScript files, and any supplemental files be routinely copied to the network drives once an application has been moved to production. This helps ensure continued operation, maintenance and support of applications during employee absence and provides an appropriate means to share code between developers. Storing source files on the network also helps ensure valuable source code is routinely backed up and could help with recovery efforts should individual hardware failures occur.

References and Acknowledgements

Teresia Arthur & Mary Jafri, "Web Enable Your SAS Applications," Proceedings of the 28th Annual SAS Users Group International Conference (2003), Paper 35-28.

Kirby Cossey, Olin Davis, and Tom Winn, "Using SAS/IntrNet to Generate Data Products from a Database: The SAO Online USAS Guide," Proceedings of SCSUG 2004, pp. 113-125 (sponsored by the South-Central SAS Users' Group, on November 7-9, 2004).

Keith Cranford & Dan Hammarstrom, "Tricks and Tips with SAS/IntrNet," Proceedings of SCSUG 2003, pp. 70-83 (sponsored by South-Central SAS Users' Group, on October 26-28, 2003).

Kevin Davidson, "Using SAS/IntrNet Software," SSU 2001 Proceedings, pp. 204-210 (sponsored by the SouthEast SAS Users Group, and the South-Central SAS Users' Group, on August 19-22, 2001)

Frank C. Dilorio, "The SAS Debugging Primer", Proceedings of the 24th Annual SAS Users Group International Conference (1999), Paper 68-24.

- Matthew Grover, "SAS Solutions for the Web: Static and Dynamic Alternatives," Proceedings of SCSUG 2003, pp. 167-178 (sponsored by South-Central SAS Users' Group, on October 26-28, 2003).
- Lauren Haworth, "HTML for the SAS Programmer," Proceedings of the 26th Annual SAS Users Group International Conference (2000), pp. 235-255.
- Lauren E. Haworth, "HTML Output," Chapter 3 in Output Delivery System: The Basics, Cary, NC: SAS Institute Inc., 2001.
- Doyle McDonald & Chas Webb, "SAS/IntrNet Configuration: Deploying a Web-Enabled SAS Environment," Proceedings of the Louisiana SAS Users Conference, pp. 77-140 (sponsored by South-Central SAS Users' Group, on June 16, 2003).
- Frederick E Pratter, Web Development with SAS by Example, Cary, NC: SAS Institute Inc., 2003.
- SAS Institute Inc., SAS/IntrNet® 9.1: Overview, (Cary, NC: SAS Institute Inc., 2005); (available online at http://support.sas.com/documentation/onlinedoc/91pdf/sasdoc_91/intrnet_overview_9281.pdf).
- SAS Institute Inc., SAS/IntrNet® 9.1: Application Dispatcher, Second Edition, (Cary, NC: SAS Institute Inc., 2005); (available online at http://support.sas.com/documentation/onlinedoc/91pdf/sasdoc_91/intrnet_dispatch_9467.pdf).
- SAS Institute Inc., SAS/IntrNet® 9.1: htmSQL, (Cary, NC: SAS Institute Inc., 2005); (available online at http://support.sas.com/documentation/onlinedoc/91pdf/sasdoc_91/intrnet_htmSQL_7023.pdf).
- SAS Institute Inc., SAS® Web Tools: SAS/IntrNet® Administration Course Notes, (Cary, NC: SAS Institute Inc., 2001).
- SAS Institute Inc., SAS® Web Tools: Static and Dynamic Solutions Using SAS/IntrNet® Software Course Notes, (Cary, NC: SAS Institute Inc., 2001).
- SAS Institute Inc., SAS® Web Tools: Advanced Dynamic Solutions Using SAS/IntrNet® Software Course Notes, (Cary, NC: SAS Institute Inc., 2001).
- Texas State Auditor's Office, "Information Systems Support Team Standards and Guidelines for Application Development", March 3, 2006,

Texas State Auditor's Office, "Standards and Guidelines for Development of SAS/IntrNet Applications", April 20, 2006,

Thomas J. Winn, Jr. "Guidelines for Coding of SAS Programs", Proceedings of SCSUG 2003, pp. 320-325 (sponsored by the South-Central SAS® Users' Group, on October 26-28, 2003).

Thomas J. Winn, Jr. "Guidelines for Coding of SAS Programs", Proceedings of the 29th Annual Conference of the SAS® Users Group International (2004), Paper 258-29.

Thomas J. Winn, Jr. "State Agency Personnel Information Reports: a SAS/IntrNet® Project", Proceedings of SCSUG 2005, pp. 343-370 (sponsored by the South-Central SAS® Users' Group, on October 16-18, 2005).

Thomas J. Winn, Jr. "State Agency Personnel Information Reports: a SAS/IntrNet® Project", Proceedings of the 31st Annual Conference of the SAS® Users Group International (2006), Paper 178-31.

The SAS/IntrNet Development Team

Special acknowledgement is given to the members of the SAS/IntrNet Development Team at the State Auditor's Office, who contributed their time and efforts during the creation of SAS/IntrNet Standards and Guidelines for our state agency: Kirby Cossey, Olin Davis, Joyce Inman, and Tom Winn.

Contact Information

Your comments and questions are valued and encouraged. You may contact the author at:

Tom Winn, Ph.D.
Senior Systems Analyst
Texas State Auditor's Office
P.O. Box 12067
Austin, TX 78711-2067
Phone: 512 / 936-9735
E-mail: twinn@sao.state.tx.us

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.