

Developing SAS/AF® Class Libraries for Automated Program Generation

Carl R. Haske, Ph.D., STATPROBE, Inc., Ann Arbor, Michigan

ABSTRACT

STATPROBE, Inc., has developed a standard SAS® programming system that has been used for a number of years to produce computer-generated reports for clinical trials. It is set up to maximize flexibility and efficiency during the course of report production for data analysis, whether the report is a basic clinical report, a report for a Data Safety Monitoring Board (DSMB), a new drug application (NDA), or integrated summaries of safety or efficacy (ISS/ISE).

The programming system consists of a generic macro library and a standard file structure that is created for each project. The standard file structure is organized hierarchically, allowing support for multiple protocols and the opportunity to develop and store reusable program code.

The final phase of development in this programming system consisted of classifying the computer-generated report specifications and creating a class library in SAS/AF corresponding to these classifications. The class library permitted the development of a user interface, linked to a program generator, to automate and standardize the production of much of the programming necessary to support a clinical report. This paper describes the class library and the program generator.

INTRODUCTION

Reports for clinical trials submitted to the Food and Drug Administration contain appendixes, consisting of very typical kinds of computer-generated reports and analyses of data. A large portion of these computer-generated reports can be classified by scope and nature of the information being reported. The data and programs to support clinical trials, although not standardized across the industry, follow certain characteristic patterns that enable the production of a class library that will in turn support the production of computer-generated reports.

Another factor to consider in the production of a set of computer-generated tables is project timing. A program may go through several phases in its life cycle, including report specifications (table shells not populated with data), interim reports, and final reports. The phases of a given program can also be classified according to several levels of completion.

One approach to implementing such a class library is to use standard macros. A macro library offers many advantages by allowing the reuse of standard segments of code to report and analyze data. In the SAS system, however, there is no direct mechanism to associate macros and class entries in a SAS catalog. Therefore, to develop a true object-oriented clinical reporting system under SAS, it is desirable to augment standard macros with a SAS/AF class library, consisting of nondisplayed class entries that classify various types of programs.

The development of the class library at STATPROBE was preceded by the development of a standard macro library and standard programming system that has been used to assist in the production of over 130 clinical reports over the past five years. The standard programming system is the foundation of the class library. The first section of this paper presents an overview of the standard programming system that was in production before the class library was developed. The next section describes the class library framework. Three sections then describe the composition of the base classes. The next section discusses data structures that are assembled in an associated graphical user interface and passed to the program generator. The final section

briefly discusses a SAS/AF application that has been developed by STATPROBE to automate and standardize the production of report specifications. Concluding remarks follow.

STANDARD PROGRAMMING SYSTEM

The standard programming system begins with the standard macro library, in which over 100 general-purpose macros are stored. These utility macros perform a wide spectrum of functions. Some macros are completely generic in function and other macros are clinical in orientation. For example, there are macros that

- Calculate age by a standard algorithm based on a birth date and an observation date.
- Determine the existence of a data set or variable.
- Perform cleanup functions by deleting temporary data sets.
- Output a standard header for reports.

Other macros produce more complicated output related to clinical data, such as:

- Summaries of adverse event and medication data.
- Comparative analyses of efficacy data.
- Categorical summaries of demographic data.
- Area under the curve (AUC) calculations.

The standard macro library is an integral component of a highly standardized and flexible programming system. The file structure is hierarchical and incorporates a client macro library and product macro library. A product may have multiple studies or protocols, each with an associated clinical database. The clinical database is at the lowest level of the hierarchy, along with a project macro library, a project database, and a report program library. The hierarchy of the system is illustrated in diagram 1.

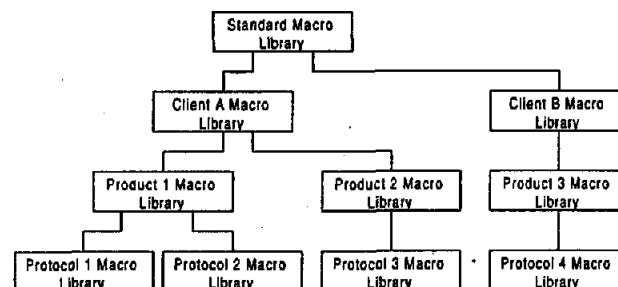


Diagram 1. Macro library hierarchy

The client macro library contains procedures that correspond to a customer's standards and are common to all projects for that customer. For example, a customer may have specific requirements regarding margins, fonts, and formatting of all computer-generated reports. A set of macros to produce report headers and footers that conform to formatting requirements would be stored in the client macro library.

The product macros contain procedures that are common to all studies for a product. Typically, different study databases for the same product have overlapping characteristics. For example, they often share data processing or manipulation operations. These operations would be embodied in macros stored in the product macro library.

Finally, the program macro library contains more specialized macros. A macro may contain a study time point schedule specification or specific database edits for study data.

The architecture of the system is an object-oriented approach of the macro implementation, allowing for inheritance and overriding using the proper search order of directories in the SASAUTOS option. For example, there are standard macros to count denominators, assign treatment groups, and calculate study day. The system allows overriding of any of these macros at the client level, product level, or project level. This is a subtle but important feature, because databases designed according to the STATPROBE standard can use the most generic macros. Conversely, databases conforming to client standards can use macros that override the standard macros. Moreover, databases for different products or projects of the same client may each conform to a different standard, so that the client macros must be overridden.

The hierarchy of the system encourages the programmer to formulate macros that are as generic as possible without sacrificing readability or maintainability. STATPROBE has used this system to develop a comprehensive set of macros and techniques that promote efficiency in our projects.

The programming structure and style used at STATPROBE is highly standardized. Most programs have common elements that can be abstracted into basic code segments. Diagram 2 illustrates a typical program structure.

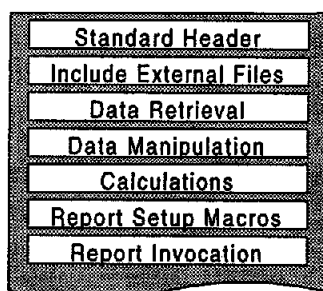


Diagram 2. Standard program components

This standard organization permits classification of programming elements. Each element corresponds to a code segment classification, and each element can be recursively reduced to a collection of classified elements until the lowest order class, that of program statement, is reached.

CLASS LIBRARY ARCHITECTURE

Classes in SAS/AF consist primarily of graphical objects or visual classes used to build a user interface. All class libraries discussed in this paper consist of nonvisual classes. Nonvisual classes are instantiated and used at run time. The simplest example of a nonvisual class is the SAS/AF object class. The object class is the parent for all other classes.

A basic example of a nonvisual subclass is the boolean class. The boolean class is the simplest example of an entry in STATPROBE's SAS/AF class libraries. It is part of a generic foundation library of classes used by applications. The boolean class is a child of the object class. It has one additional instance variable called *value*, that has a default value of 1, and three additional methods described in table 1. The code for the boolean methods is displayed in listing 1.

Table 1. Methods for Boolean Class

Method	Description
--------	-------------

GET_VALUE	Retrieves the value of the boolean object.
SET_VALUE	Sets the value of the boolean object.
TOGGLE	Toggles the value of the boolean object.

```

getvalue:
  Method val 8;
  val=value;
  Endmethod;

setvalue:
  Method val 8;
  If val in (0,1) then
    value=val;
  Endmethod;

toggle:
  Method;
  value = not value;
  Endmethod;

```

Listing 1. Methods for BOOLEAN class

A boolean object has many potential uses in a SAS/AF system. It is commonly used to propagate a mode throughout an application at run time. For example, a well-designed application will have a large volume of code integrated in the system to both debug and test the system. A developer can use boolean objects to track debug and test modes, altering modes as needed at run time. These debug objects can be inserted into the global application list and are available during run time.

SAS programs that report clinical data can be classified by type of program, such as listings, summaries, analyses, and graphs. Programs can be further classified by the type of data being reported, such as demographics, adverse events, vital signs, and medications. Within each class of program, the type of data processing and program function can, to a large part, be categorized.

STATPROBE has developed a class library consisting of nonvisual classes to support the automatic generation of clinical programming. The class library has three constructs used as base classes. The three levels of classes are:

- Program
- Code segment
- Statement

Diagram 3 illustrates the organization of the class library.

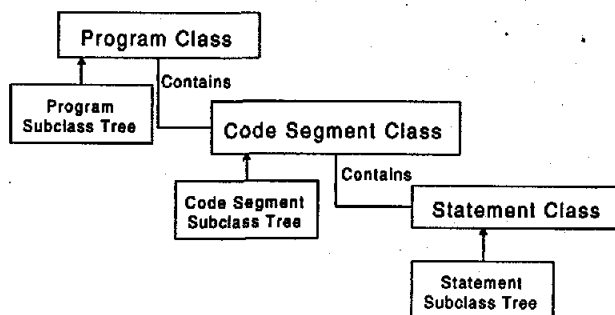


Diagram 3. Class library schematization

A program object consists of an ordered set of code segment objects, and a code segment object consists of an ordered set of statement objects.

A program object is responsible for input to and output from its associated physical file. It is also responsible for being aware of its code segment objects and communicating data and commands to them. A code segment object is responsible for maintaining its statements and writing its statements to its parent program. A statement object corresponds to a SAS program statement and is responsible for writing its contents to its parent code segment.

A program object is instantiated when the user interface passes data about the program to the program generator. The data for a program object are stored in a SCL list structure. The items of the SCL list are SCL lists with data corresponding to the various classes of code segments contained in the program. The SCL list for each code segment is a SCL list corresponding to code segments statements. The structure of the class library allows for parallel and independent development of the user interface and the class library. Adding a new standard element to the global programming system stimulates the development of new classes in the class library, which in turn impels expansion of the user interface.

PROGRAM CLASS

The program class is the base class for all types of SAS programs. The parent class for the program class is the SAS External File Class. The program class inherits all instance variables and methods from the external file class.

STATPROBE's standard system includes these program categories:

- Listings
- Summaries
- Analyses
- Graphs
- Format library
- Database generator
- Ad hoc

There are further classifications of programs based on the type of data being reported. With the exception of the ad hoc class, programs in each category have a similar design or program flow.

Table 2 shows the instance variables for the program class. Two instance variables are needed for the parent class of all program classes: a file identifier, and an object list to track all of a program object's code segments.

Table 2. Program Class Instance Variables

Instance Variable	Description
file_id	File handle for the program physical file.
oblist	List handle for the program code segments.

Table 3 shows the methods in the program base class. The program methods take care of managing a program object's associated file and a program object's code segment objects. Listing 2 shows the code for the GET_FILENAME method. The method gets the first object in the program, which is a standard header containing the program file name. The file name is returned in the method parameter *fn*.

Listing 3 shows the WRITE_PROGRAM method. The method opens the program file, sends the message to each code segment to write itself, and then closes the file. Subclasses of the program class add instance variables and methods as needed.

For the program class to be used, the following steps must occur sequentially:

- Instantiate and setup a program object.
- Set the data of the program object.
- Tell the program object to write the associated program file.

A data structure manifest as a SCL list must be constructed and used to set the data of a program object. The SCL list consists of named items that correspond to code segment objects. The code segment class is described in the next section. The construction of the SCL list is typically achieved by a user interface. The user interface collects user input relating to the content of the desired report and translates the input into data stored in the SCL list.

Table 3. Program Class Methods

Method	Description
GET_FILENAME	Retrieves the file specification associated with the program object.
GET_OBJECT	Retrieves a files code segment object based on an index.
PROGRAM_CLOSE	Closes the program object's associated file.
PROGRAM_OPEN	Opens the program object's associated file.
READ_LINE	Reads a text line from the program object's associated file buffer.
SETUP_FILE	Associates a SAS fileref to the program object's associated file.
SET_DATA	Associates the data contained in an SCL list with the program object's oblist instance variable.
WRITE_LINE	Writes a text line from the program object's associated file buffer.
WRITE_PROGRAM	Writes the program object's code segments to the program object's associated file buffer.

```

*** GET_FILENAME ***;
GET_FN:
  Method fn $;
  call send(_self_,'GET_OBJECT',1,obid);
  If nameditem(getniteml(obid,'DATA'),
    'FILENAME') then
    fn=getnitemc(getniteml(obid,
      'DATA'),'FILENAME');
  Endmethod;

```

Listing 2. GET_FILENAME method

```

*** WRITE_PROGRAM ***;
WPROGRAM:
  Method;
  call send(_self_,'PROGRAM_OPEN','O');
  Do i=1 to listlen(oblist);
    call send(getiteml(oblist,i),'WRITE');
  End;
  call send(_self_,'PROGRAM_CLOSE');
  Endmethod;

```

Listing 3. WRITE_PROGRAM method

CODE SEGMENT CLASS

The code segment class represents the high-level components of a program. A code segment object is an ordered set of program statements. Some examples of code segment classifications in a SAS program are

- Comment block
- Macro definitions
- Data procedures
- Procedure blocks

The code segment class provides the basis for these and other classified portions of SAS code.

Table 4. Code Segment Class Instance Variables

Instance Variable	Description
data	List handle for the code segment input data.
lines	List handle for the code segment statements.
program	Numeric list identifier for the parent program object.

Table 4 shows the instance variables for the code segment class. The instance variables for the base class include a SCL list of data for the code segment, a SCL list of data for the code segment statements, and a numeric identifier for the parent program object of the code segment.

Table 5. Code Segment Class Methods

Method	Description
ADD_LINE	Add a statement object to the code segment object.
GET_PROGRAM	Get the identifier for the code segment object's parent program object.
SET_DATA	Set the data instance variable.
SET_LINES	Initialize the lines instance variable using input data.
SET_PROGRAM	Set the program instance variable.
WRITE	Write the code segment to the parent program object's associated file.
WRITE_BLANK	Write a blank line to the parent program object's associated file.

Table 5 shows the methods for the base code segment class. The methods manage the code segment statements, manage the parent object identifier, and write the code segment to the parent program object's associated file. The methods allow the code segment to send and receive messages from the parent program object. The *set_program* method allows an application to assign the parent program object to a code segment. The *get_program* method allows a code segment object to retrieve the object identifier of the parent program object. The remaining methods are used by the parent program object to instruct the code segment object to perform an action such as adding a statement or writing the program statements to an associated physical file.

Listing 4 shows the ADD_LINE method. This method adds a statement object to a code segment object. The method instantiates a new statement object, assigns the parent code segment to the new statement object, sets the text of the statement object, and finally tracks the line number of the statement.

```
*** ADD_LINE ***;
ADD_LINE:
Method class $ line 8 optional= text $ indent 8;
clid=loadclass('aaa.generate.'llclassll'.class');
obid=instance(clid);
call send(obid,'SET_CODE',_self_);
If text^=_blank_ then
  call send(obid,'SET_TEXT',text,indent);
```

```
lines=insertl(lines,obid,line,'LINE'llline);
Endmethod;
```

Listing 4. ADD_LINE method

The code segment class is used to construct a meaningful program object. A nontrivial program object consists of an ordered list of objects derived from the code segment class. To use the code segment class, a program object reads the program data that a user interface passes. These data contain the type of code segments that make up the program. For each element of data, the program instantiates the proper code segment object and sets the data of the code segment object; then the code segment object is inserted into the program object's object list.

STATEMENT CLASS

The classification of SAS program statements is extensive and complicated because of the enormous size of the SAS system. This class library has the highest level of maintenance. Types of SAS statements include:

- Include statements
- Macro invocations
- Comments
- Data step statements
- Set statements
- Merge statements
- Proc statements

There are many other classes of SAS statements, too numerous to mention. The statement class is the base class for all SAS statement classes. Table 6 shows the instance variables for the base statement class. There are variables to identify the parent code segment, the maximum length of the statement text, and the text of the statement.

Table 6. Statement Class Instance Variables

Instance Variable	Description
codeseg	An identifier for the statement object's parent code segment object.
linelen	A numeric variable for the maximum character length of a program statement.
text	The text of the statement object.

Table 7 shows the methods for the base statement class. The methods set the parent code segment, statement text, and write the statement.

Table 7. Statement Class Methods

Method	Description
SET_CODE	Sets the statement object's parent code segment.
SET_TEXT	Sets the statement object's text.
WRITE	Writes the statement object's text to the parent program object of the statement object's parent code segment.

Listing 5 shows the WRITE method for the statement class. The statement object gets the identifier for the program object to which the statement belongs and then sends to the program the message to write the statement's text to the program.

```
*** WRITE ***;
WRITE:
Method;
call send(codeseg,'GET_PROGRAM',program);
```

```
call send(program,'WRITE_LINE',text);
Endmethod;
```

Listing 5. WRITE method

The statement class is used by the code segment class. A code segment is composed of an ordered list of statements or lines. A line is actually a type of statement object. When a program object's data are passed by a user interface, code segment objects are instantiated and, in turn, various types of statement objects are instantiated to constitute the code segment object.

DATA STRUCTURE

Each level of messaging between the various objects in this system is trivial; however, the resulting data structure for a program object can be quite complicated and cumbersome. Listing 6 shows a portion of data that is collected by a user interface and passed to the program object via the *set_data* method. The complete listing appears in appendix 1. This portion of the data defines the basic information in a program header. The list identified by address [9] is an embedded list of lines that describe the program purpose. The remainder of the list items are some of the standard documentation elements in STATPROBE programs.

```
PHEADER=(
CLIENT='ACME DRUGS'
PROJECT='HEAL-ME-FAST'
FILENAME='TEST1.SAS'
CREATOR='CRH'
PARENT='NONE'
INPUT='NONE'
OUTPUT='NONE'
PURPOSE=(
PURPOSE1='This is a test program'
PURPOSE2='It tests the program class and'
PURPOSE3='its associated file.')[9]
)[7]
```

Listing 6. List structure of program header data passed by a user interface

The class entry for the program header class is *pheader.class*; thus the method that reads the data from the user interface uses the name of a list item to determine which code segment class to instantiate. If the list item name does not correspond to a code segment class, then the program generator ignores the list item. Listing 7 shows the code segment object that is instantiated from the user interface data in listing 5 and inserted in a program's object list. This is one code segment of a complete program object that is listed in appendix 2. The resulting program appears in appendix 3.

```
OBJECT1=(
CLASS_=91
DESC='Program header code segment class'
DATA=(
CLIENT='ACME DRUGS'
PROJECT='HEAL-ME-FAST'
FILENAME='TEST1.SAS'
CREATOR='CRH'
PARENT='NONE'
INPUT='NONE'
OUTPUT='NONE'
PURPOSE=(PURPOSE1='This is a test program'
PURPOSE2='It tests the program class and'
PURPOSE3='its associated file.')[123]
)[115]
```

Listing 7. List structure of a program header object in a program object list

AN APPLICATION

STATPROBE has developed a SAS/AF application, FERPS (Front End for developing Report Program Skeletons,) that uses the program generator. The system is designed to support multiple template libraries for several clients. The application provides a GUI to design reports, thus eliminating the need for programmer intervention in the process of specification development. Details of the system are described in Hamilton and Lester (1996). The system acts as a type of wizard, sequentially collecting the user requirements, allowing preview of the report output, and generating the underlying SAS program.

CONCLUSION

The nonvisual classes used in the program generator system allow the system to be independent of the user interface platform. Currently a web-based user interface system is under development to capture the report parameters, build the data structure, initiate SAS on a server and pass program data to the program generator, and retrieve the report results in HTML format for display to the user.

REFERENCES

- Haske, Carl R. (1996), "Application Development Templates in SAS/AF[®]," *Proceedings of the Twenty-Second Annual SAS[®] Users Group International Conference*, 7-12, Cary, NC: SAS Institute Inc..
- Haske, Carl R. (1996), "Using SAS/AF[®] for Managing Clinical Data," *Proceedings of the Twenty-Second Annual SAS[®] Users Group International Conference*, 557-562, Cary, NC: SAS Institute Inc..
- Hamilton, Jeff, and Jennifer Lester (1996), "Using SAS/AF[®] as a Front End for Developing Report Program Skeletons," *Proceedings of the Twenty-Second Annual SAS[®] Users Group International Conference*, 74-77, Cary, NC: SAS Institute Inc..
- SAS Institute, Inc. (1993), *SAS/AF Software: FRAME Entry, Usage and Reference, Version 6, First Edition*, Cary, NC: SAS Institute Inc.
- SAS Institute, Inc. (1994), *SAS Screen Control Language: Reference, Version 6, Second Edition*, Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

SAS and SAS/AF are registered trademarks of SAS Institute Inc. in the USA and other countries. * indicates USA registration.

The author thanks Jeff Hamilton and Jennifer Lester for generously offering their permission to discuss the FERPS application. Also, thanks to Paul Schwankl for assistance in the preparation of this paper.

AUTHOR'S ADDRESS

Carl R. Haske, Ph.D.
STATPROBE, Inc.
3885 Research Park Drive
Ann Arbor, MI 48108
(313) 769-5000 x115
E-mail: chaske@statprobe.com

APPENDIX 1

Data Collected and Passed by User Interface

```
Program Data =
( PHEADER= ( CLIENT='ACME DRUGS'
PROJECT='HEAL-ME-FAST'
FILENAME='TEST1.SAS'
CREATOR='GRH'
PARENT='NONE'
INPUT='NONE'
OUTPUT='NONE'
PURPOSE=( PURPOSE1='This is a test program'
PURPOSE2='It tests the program class and'
PURPOSE3='its associated file.'
))19)
))17)
INCLUDE=( ENVIRON='H:\DATA\SAS\ACME\H001-002\HMF0102\HMF12PM\ENVIRON.MAC'
LISTTOOL='H:\DATA\SAS\ACME\H001-002\HMF0102\HMF12ML\LISTTOOL.MAC'
))11)
RETRIEVE= ((113)
COLWID=( W1=15
W2=14
W3=12
W4=25
W5=25
W6=10
W7=20
))15)
MACROPUT=( NAME='header'
L1=( 1='CENTER NUMBER'
))10)
))17)
MACROPUT=( NAME='colhead'
L1=( 3='DATE'
7='RELATIONSHIP'
))23)
L2=( 1='TREATMENT'
2='PATIENT'
3='ADVERSE'
4='ONSET DATE'
5='RESOLVED'
7='TO STUDY'
))25)
L3=( 1='GROUP'
2='NUMBER'
3='EVENT'
4='(STUDY DAY)'
5='(STUDY DAY)'
6='SEVERITY'
7='DRUG'
))27)
))21)
MACROPUT=( NAME='body'
))20)
MACROPUT=( NAME='footer'
L1=( 1='SERIOUS ADVERSE EVENT'
))30)
))31)
CODESEG=( 'report(dsn=
,
softvars=
,
header=header,
subhead=colhead,
body=body,
foot=footer,
tblnum=1);'
))35)
))5)
```

APPENDIX 2

Data for a Program Object

```
( _CLASS_='57
DESC='Program class: BASE CLASS'
FILE_ID=1000
OBLIST=( OBJECT1=( _CLASS_='91
DESC='Program header code segment class'
DATA=( CLIENT='ACME DRUGS'
PROJECT='HEAL-ME-FAST'
FILENAME='TEST1.SAS'
CREATOR='GRH'
PARENT='NONE'
INPUT='NONE'
OUTPUT='NONE'
PURPOSE=( PURPOSE1='This is a test program'
PURPOSE2='It tests the program class and'
PURPOSE3='its associated file.'
))123)
))115)
LINES=( LINE1=( _CLASS_='121
DESC='STARLINE.CLASS'
CODESEG=(...))113)
LINELEN=76
TEXT=*****
))141)
LINE2=( _CLASS_='147
DESC='COMMENT.CLASS'
CODESEG=(...))113)
LINELEN=76
TEXT=**** Specification Section
****
))155)
LINES=( _CLASS_='121
DESC='STARLINE.CLASS'
CODESEG=(...))113)
LINELEN=76
TEXT=*****
))161)
LINE4=( _CLASS_='147
DESC='COMMENT.CLASS'
CODESEG=(...))113)
LINELEN=76
TEXT=**** Physical Specifications
****
))185)
LINES=( _CLASS_='147
DESC='COMMENT.CLASS'
CODESEG=(...))113)
LINELEN=76
TEXT=**** Project ID: ACME DRUGS HEAL-ME-FAST
****
))169)
LINES=( _CLASS_='147
DESC='COMMENT.CLASS'
CODESEG=(...))113)
LINELEN=76
TEXT=**** Program Name: TEST1.SAS
****
))173)
LINE7=( _CLASS_='147
DESC='COMMENT.CLASS'
CODESEG=(...))113)
LINELEN=76
TEXT=**** Created: 03/30/97 14:10
****
))177)
LINES=( _CLASS_='147
DESC='COMMENT.CLASS'
CODESEG=(...))113)
LINELEN=76
TEXT=**** By: GRH
****
))181)
LINES=( _CLASS_='147
DESC='COMMENT.CLASS'
CODESEG=(...))113)
LINELEN=76
TEXT=**** Derived From: NONE
****
))185)
LINE10=( _CLASS_='147
DESC='COMMENT.CLASS'
CODESEG=(...))113)
LINELEN=76
TEXT=**** Input Data: NONE
****
))189)
LINE11=( _CLASS_='147
DESC='COMMENT.CLASS'
CODESEG=(...))113)
LINELEN=76
TEXT=**** Output Data: NONE
****)
```



```

      ][409]
      LINE11=( _CLASS_131
      DESC='Program statement class: BASE CLASS'
      CODESEG=(...)[355]
      LINELEN=76
      TEXT='      @&c2 'NUMBER''
      ][407]
      LINE12=( _CLASS_131
      DESC='Program statement class: BASE CLASS'
      CODESEG=(...)[355]
      LINELEN=76
      TEXT='      @&c3 'EVENT''
      ][411]
      LINE13=( _CLASS_131
      DESC='Program statement class: BASE CLASS'
      CODESEG=(...)[355]
      LINELEN=76
      TEXT='      @&c4 '(STUDY DAY)''
      ][415]
      LINE14=( _CLASS_131
      DESC='Program statement class: BASE CLASS'
      CODESEG=(...)[355]
      LINELEN=76
      TEXT='      @&c5 '(STUDY DAY)''
      ][419]
      LINE15=( _CLASS_131
      DESC='Program statement class: BASE CLASS'
      CODESEG=(...)[355]
      LINELEN=76
      TEXT='      @&c6 'SEVERITY''
      ][423]
      LINE16=( _CLASS_131
      DESC='Program statement class: BASE CLASS'
      CODESEG=(...)[355]
      LINELEN=76
      TEXT='      @&c7 'DRUG''
      ][427]
      LINE17=( _CLASS_131
      DESC='Program statement class: BASE CLASS'
      CODESEG=(...)[355]
      LINELEN=76
      TEXT='Mend colhead;'
      ][431]
    ][359]
    PROGRAM=01
    ][355]
  OBJECT7=( _CLASS_323
  DESC='MACROPUT.CLASS'
  DATA=( NAME='body'
  ][437]
  LINES=( LINE1=( _CLASS_131
  DESC='Program statement class: BASE CLASS'
  CODESEG=(...)[435]
  LINELEN=76
  TEXT='Macro body;'
  ][441]
  LINE2=( _CLASS_131
  DESC='Program statement class: BASE CLASS'
  CODESEG=(...)[435]
  LINELEN=76
  TEXT='Mend body;'
  ][445]
  ][439]
  PROGRAM=01
  ][435]
  OBJECT8=( _CLASS_323
  DESC='MACROPUT.CLASS'
  DATA=( NAME='footer'
  L1=( L1='SERIOUS ADVERSE EVENT'
  ][457]
  ][451]
  LINE9=( LINE1=( _CLASS_131
  DESC='Program statement class: BASE CLASS'
  CODESEG=(...)[449]
  LINELEN=76
  TEXT='Macro footer;'
  ][455]
  LINE2=( _CLASS_131
  DESC='Program statement class: BASE CLASS'
  CODESEG=(...)[449]
  LINELEN=76
  TEXT='Put @&c1 'SERIOUS ADVERSE EVENT''
  ][461]
  LINE3=( _CLASS_131
  DESC='Program statement class: BASE CLASS'
  CODESEG=(...)[449]
  LINELEN=76
  TEXT='Mend footer;'
  ][465]
  ][453]
  PROGRAM=01
  ][449]
  OBJECT9=( _CLASS_101
  DESC='Code segment class: BASE CLASS'
  DATA=( 'report(dsn',
  sortvar=,
  header=header,
  subhead=colhead,
  body=body,
  foot=footer,
  tblnum=1);
  ][471]
  LINES=( LINE1=( _CLASS_131
  DESC='Program statement class: BASE CLASS'
  CODESEG=(...)[469]
  LINELEN=76
  TEXT='report(dsn',
  ][477]
  LINE2=( _CLASS_131
  DESC='Program statement class: BASE CLASS'
  CODESEG=(...)[469]
  LINELEN=76
  TEXT='sortvar=,'
  ][481]
  LINES=( _CLASS_131

```

```

      DESC='Program statement class: BASE CLASS'
      CODESEG=(...)[469]
      LINELEN=76
      TEXT='      header=header,'
      ][485]
      LINE4=( _CLASS_131
      DESC='Program statement class: BASE CLASS'
      CODESEG=(...)[469]
      LINELEN=76
      TEXT='      subhead=colhead,'
      ][489]
      LINE5=( _CLASS_131
      DESC='Program statement class: BASE CLASS'
      CODESEG=(...)[469]
      LINELEN=76
      TEXT='      body=body,'
      ][493]
      LINE6=( _CLASS_131
      DESC='Program statement class: BASE CLASS'
      CODESEG=(...)[469]
      LINELEN=76
      TEXT='      foot=footer'
      ][497]
      LINE7=( _CLASS_131
      DESC='Program statement class: BASE CLASS'
      CODESEG=(...)[469]
      LINELEN=76
      TEXT='      tblnum=1);'
      ][501]
    ][473]
    PROGRAM=01
    ][469]
  ][83]
  PATH=( 'C:\CARL\TEMP\TEST1.SAS'
  ][89]
  ATTRIBUTES=( FILENAME='C:\CARL\TEMP\TEST1.SAS'
  RECFM='V'
  LRECL='256'
  ][87]
  NAME='MYFILE'
  ENGINE=' '
  TYPE='FILE'
  CASE='M'
  ][81]

```

APPENDIX 3

The Resulting Program

```
*****
*** Specification Section ***
*****
*** Physical Specifications ***
*** Project ID: ACME DRUGS HEAL-ME-FAST ***
*** Program Name: TEST1.SAS ***
*** Created: 07/03/97 15:47 ***
*** By: CRH ***
*** Derived From: NONE ***
*** Input Data: NONE ***
*** Output Data: NONE ***
*** Operating Sys: WIN ***
*** Language: SAS, version 6.12 ***
*** Logical Specifications ***
*** Purpose: This is a test program ***
*** It tests the program class and ***
*** its associated file. ***
*****

*** Include external macro files ***
%include "H:\DATA\SAS\ACME\H001-002\HMF0102\HMF12PGM\ENVIRON.MAC";
%include "H:\DATA\SAS\ACME\H001-002\HMF0102\HMF12ML\LISTTOOL.MAC";

*** Retrieve report title and header information ***
%let pgmname=TEST1.SAS;
%datapgm(&pgmname);

*** Set the report column widths ***
%colwids(15,14,12,25,25,10,20);

%Macro header;
  Put @&c1 'CENTER NUMBER'
%Mend header;

%Macro colhead;
  Put @&c5 'DATE'
    @&c7 'RELATIONSHIP'
    / @&c1 'TREATMENT'
      @&c2 'PATIENT'
      @&c3 'ADVERSE'
      @&c4 'ONSET DATE'
      @&c5 'RESOLVED'
      @&c7 'TO STUDY'
    / @&c1 'GROUP'
      @&c2 'NUMBER'
      @&c3 'EVENT'
      @&c4 '{STUDY DAY}'
      @&c5 '{STUDY DAY}'
      @&c6 'SEVERITY'
      @&c7 'DRUG'
%Mend colhead;

%Macro body;
%Mend body;

%Macro footer;
  Put @&c1 '* SERIOUS ADVERSE EVENT'
%Mend footer;

%report(dsn=,
  sortvars=,
  header=header,
  subhead=colhead,
  body=body,
  foot=footer,
  tblnum=1);
```