A TOOLBOX APPROACH TO NETWORKING SAS® EXPERTS
OR
RE-USING INSTEAD OF RE-INVENTING THE WHEEL
Don Henderson, ORI, Inc.

1. ABSTRACT

Toolboxes are often used by data processing groups in the development of computer systems. The concept of a toolbox is straightforward; generalized tools (e.g., software modules) are developed and packaged so that they may be easily used by an entire team or staff of people. PROC PRINT is an excellent example. SAS programmers do not have to write PUT and FILE statements each time they want to display a report of data; SAS Institute has developed a tool (PROC PRINT) and put it in a toolbox (the procedure library) available to all SAS users. Macro tools presented at this and previous SAS conferences provide other examples.

Unfortunately, toolboxes have traditionally been used only for generic routines which can be taken "off the shelf" and used as is (e.g., the SAS procedure library and user developed libraries of procedures and Macros). Toolboxes have a much broader application, however, especially in the educational and consulting environment. By generalizing toolboxes to include "sample applications" which can be used in a variety of application areas, it is possible to greatly enhance and extend the ability and productivity of SAS programmers.

2. INTRODUCTION

A toolbox is a collection of routines or software modules that an applications programmer or systems designer can use or consult in the development of new programs or systems. Programs in a toolbox can range on a continuum from the SAS procedure libraries, through user developed generalized Macro routines, to sample or prototype code which toolbox users can use to guide them in solving their own special problems.

Toolboxes are widely used by data processing organizations, both at the organizational and application area level in the form of a library of routines or subroutines which are available for general use. This is true regardless of the language of choice. In the more traditional languages such as PL/I or COBOL, such toolboxes contain subroutines. The SAS System itself has taken advantage of this concept: it is a programming language (the DATA Step) with a built-in collection of tools (procedure library) that are automatically available to the user. In fact, other Institute products are nothing more than a second level of sample toolboxes which toolbox users can use to guide them in solving their own special problems.

Toolboxes are widely used by data processing organizations, both at the organizational and application area level in the form of a library of routines or subroutines which are available for general use. This is true regardless of the language of choice. In the more traditional languages such as PL/I or COBOL, such toolboxes contain subroutines. The SAS System itself has taken advantage of this concept: it is a programming language (the DATA Step) with a built-in collection of tools (procedure library) that are automatically available to the user. In fact, other Institute products are nothing more than a second level of sample toolboxes which toolbox users can use to guide them in solving their own special problems.

There are numerous reasons why toolboxes are advantageous. Many of them are the same reasons why the SAS System itself is so popular and widely used. Among the more prominent reasons are:

- they minimize the amount of application specific code that needs to be written;
- if a common function needs to be performed in a variety of places, the use of one toolbox routine increases the reliability of the application;
- the availability of easy to use tools can lead to the development of more robust, reliable and effective systems;
- they increase the productivity and, through time, the expertise of toolbox users; and
- their use increases the probability that a project can be completed in a timely manner.

Two levels of sample toolboxes will be discussed — Macro libraries, and libraries of sample or prototype code — followed by a discussion of how to build and maintain toolboxes.

3. SAMPLE TOOLBOXES

3.1 A LIBRARY OF MACRO TOOLS

The use of Macro libraries and tools has been discussed in many previous papers (see references 1-7). Many organizations maintain their own Macro libraries because through their use, programmers and analysts are more productive and build and maintain better systems. In general, such libraries (or toolboxes) should contain packaged and easy to use code which solve common problems to the organization. These problems may be standard problems (e.g., checking for an empty transaction file) or may solve some problem or produce some specific report (e.g., produce report ONE on a specified user data set).

As mentioned above, a general Macro library should contain Macros which solve common data processing problems such as:

- conditionally producing test output based on the setting of a Macro variable so that "test code" can always be available to a production application;
- checking for the existence of a data set;
- checking to see if a data set (transaction file) has observations;
- creating a format table from information stored in a SAS data set;
- checking for the validity of user input:
  - valid SAS names,
  - variables are of correct type (numeric vs. character),
  - variables exist in a specified data set,
  - user selected option is valid.

A second level of Macro toolboxes corresponds to application-specific tools. These come into play if SAS based systems are designed as discussed in reference 5. Systems can be designed and implemented so that they are implemented through a "driver" Macro which calls or executes a series of "subroutine" Macros (see Figure 1). Each "subroutine" Macro can then be used from the toolbox. For example, suppose that system ABC produces report ONE by having the driver program call %REPTONE on a specified data set. Report ONE can be generated either from within the system...
or externally. It is only necessary to call Macro %REPTONE with the
required input parameters, e.g., %REPTONE(...,DATA=SPECIAL).

```sas
-%MACRO ABC(parameter list);
  %GLOBAL •••;
  %LOCAL •••;
  %EXTRACT (...);
  %RESHAPE (... ,OUT=FINAL);
  %REPTONE (... ,DATA=FINAL);
-%END ABC;
```

**FIGURE 1. Sample Driver Macro**

3.2 A LIBRARY OF PROTOTYPE CODE

A library of prototype code can contain a wide variety of programs. There is no judging whether the inclusion of a typical piece of code will be productive or not. Most often, code that is unique in some way, either because it solves a difficult problem or because it is a novel solution to a common problem, is the most productive type of code to include. It is important to recognize that such toolboxes are not bound by application area. In order to illustrate these points, this section will discuss, by example, the utility of toolboxes of prototype code.

The first example presents a problem that a research scientist is having with data he has collected in a blind trial. His data set (CHEM.TRIAL) contains observations for many subjects and he needs to compute descriptive statistics by treatment on a specified subset of the patients. He has two problems with the data set. First, it has no treatment identifier. Second, he only wants to select subjects who started the study after May 1986. Since the data set does not contain the necessary information for him to complete his task and because he cannot add the information to the permanent data set, he decides to create a second dataset (OTHER.DESCRIP) which has the subject identifier (SUBJ_ID) and the appropriate identification and selection fields. He develops the code and places it in the prototype code toolbox (see Figure 2).

```sas
DATA STATS;
MERGE CHEM.TRIAL OTHER.DESCRIP;
BY SUBJ_ID;
DROP START;
IF 'OlMAY86' D <= START;
RUN;
PROC MEANS;
BY TRTMNT;
TITLE 'Descriptive Statistics by Treatment';
RUN;
```

**FIGURE 2. Sample Toolbox Code**

Upon reviewing the prototype code library, the marketing analyst discovers that the solution developed by the research scientist can be used to solve her problem. First, she creates a parameter file data set REPT.GROUPING (see Figure 3) which specifies for each DEMOGRP the additional special groups to which it belongs.

<table>
<thead>
<tr>
<th>DEMOGRP</th>
<th>N_GROUPS</th>
<th>GRP1</th>
<th>GRP2</th>
<th>GRP3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>S1</td>
<td>S2</td>
<td>S3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
<td>S3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td></td>
<td>S3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>S1</td>
<td>S3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>S2</td>
<td>S3</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td></td>
<td>S1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td></td>
<td>S2</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td></td>
<td>S1</td>
<td></td>
</tr>
</tbody>
</table>

**FIGURE 3. Parameter File Data Set**

Next, she includes the code shown in Figure 4 just before the call to %SALESRPT.

```sas
DATA REPORT;
MERGE ORIGDATA REPT.GROUPING;
BY DEMOGRP;
ARRAY GRPS (3) GRP1 GRP2 GRP3;
DROP N_GROUPS GBP1 GRP2 GRP3 I;
OUTPUT; /* THE ORIGINAL GROUPS */
/* Output the record, associating it with each new */
/* group to which the input group belongs to */
DO I=1 TO N_GROUPS;
  DEMOGRP..,(,'RP8(I);
  OUTPUT; /* EACH NEW GROUP*/
END;
RUN;
%SALESRPT(DATA=REPORT, /*NOW CONTAINS NEW GROUPS*/
  ...as before...)
```

**FIGURE 4. Toolbox Code To Use the Parameter Data Set**

She generalized the solution that the research scientist came up with. His problem was to assign the group to a record. Her solution was to "duplicate" the records for specific groups, creating a new record for each of the new group identifiers.

The last example presents a problem from the accounting department. The programmer responsible for Profit and Loss (P&L) report generation has just received another change to the order and calculation formulas for the P&L account lines. He has become tired of digging into the body of the code to change the sort order and calculation for each account line so he decides to try and come up with a procedure that is not as labor intensive. Upon reviewing the prototype code library, he discovers the solution developed by the marketing analyst and decides that such a table driven approach will work for him as well. Part of the P&L with its new formulas are shown in Figure 5.

**FIGURE 5. Selected P&L Account Line Formulas**

Cross Sales: Accounts 5801, 5802 and 5905
Cost: Account 1203 plus 1/2 of 1204
Net Sales: Gross Sales - Costs
Taxes: .25 - Net Sales
Profit: Net Sales - Taxes
A partial listing of his data set that translates P&L lines to their sort order and provides labels is stored in PANDLLINES (Figure 6).

<table>
<thead>
<tr>
<th>LINE</th>
<th>SORTORDR</th>
<th>LABEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>17</td>
<td>Gross Sales</td>
</tr>
<tr>
<td>17</td>
<td>10</td>
<td>Costs</td>
</tr>
<tr>
<td>22</td>
<td>19</td>
<td>Net Sales</td>
</tr>
<tr>
<td>28</td>
<td>20</td>
<td>Taxes</td>
</tr>
<tr>
<td>34</td>
<td>21</td>
<td>Profit</td>
</tr>
</tbody>
</table>

**FIGURE 6. P&L Lines.**

A partial listing of the parameter file data set of formulas, PANDLFORMULAS is given in Figure 7.

<table>
<thead>
<tr>
<th>ACCOUNT</th>
<th>N_LINES</th>
<th>TO1</th>
<th>TO2</th>
<th>TO3</th>
<th>TO4</th>
<th>FR1</th>
<th>FR2</th>
<th>FR3</th>
<th>FR4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1203</td>
<td>4</td>
<td>17</td>
<td>22</td>
<td>28</td>
<td>34</td>
<td>1</td>
<td>-1</td>
<td>-25</td>
<td>-75</td>
</tr>
<tr>
<td>1204</td>
<td>4</td>
<td>17</td>
<td>22</td>
<td>28</td>
<td>34</td>
<td>.5</td>
<td>-.5</td>
<td>-.125</td>
<td>-.375</td>
</tr>
<tr>
<td>5801</td>
<td>4</td>
<td>14</td>
<td>22</td>
<td>28</td>
<td>34</td>
<td>1</td>
<td>1</td>
<td>-25</td>
<td>.75</td>
</tr>
<tr>
<td>5802</td>
<td>4</td>
<td>14</td>
<td>22</td>
<td>28</td>
<td>34</td>
<td>1</td>
<td>1</td>
<td>-25</td>
<td>.75</td>
</tr>
<tr>
<td>5805</td>
<td>4</td>
<td>14</td>
<td>22</td>
<td>28</td>
<td>34</td>
<td>1</td>
<td>1</td>
<td>-25</td>
<td>.75</td>
</tr>
</tbody>
</table>

**FIGURE 7. P&L Formulas.**

Looking at the variables T01-T04 and FR1-FR4, one can see that Line 14 (Gross Sales) is calculated as 5801+5802+5805; Line 22 (Net Sales = Gross Sales - Cost) is calculated as 5801+5802+5805-1203-1204/2. The calculations are now totally parameter driven.

The code for his solution to the problem is shown in Figure 8.

```sql
DATA PANDL{KEEP=LINE ORDER COST};
create formats for SORTORDR and LABEL.
formats using PANDL.LINES as input toigion code.
RUN;
Call the toolbox routine that makes account reports
the tool library, another programmer found formats for SORTORDR and LABEL.
DATA PANEL KEEP=LINE ORDER COST;
MERGE INPUT (KEEP=ACCOUNT AMOUNT)
BY ACCOUNT;
ARRAY LINES (*) TO1 TO2 TO3 TO4 ...
ARRAY FACTORS(*) FR1 FR2 FR3 FR4 ...
/* Associate each input account with the */
/* appropriate account line, including the */
/* application of any factors */
DO I=1 TO N_LINES;
  LINE=LINES(I);
  COST=AMOUNT*FACTORS(I);
  ORDER=SORTORDR(LINE, SORTORDR.);
  OUTPUT;
END;
RUN:

Call the routine/Macro that produces the P&L report, making sure to use the LABEL format created above. Note that this routine will aggregate the data to the account line level.

**FIGURE 8. Parameter Driven P&L Generation**

Once again, someone has used the solution to a simpler problem to solve a more complex one. For this solution, the P&L programmer built upon the market research analyst's solution by including factors in his parameter file. He now has an application which can absorb changes merely by changing the parameter files. It is no longer necessary to change and retest the main body of the application code.

Each one of the three toolbox members discussed above contributed to a solution of a different problem in a different application area. None of the above pieces of code lent itself to being addressed in a generalized Macro, yet their inclusion in a toolbox performed a comparable function to a Macro. In each case, since the developer of the code included it in a toolbox library, another programmer found a solution to his/her own problem. It is unlikely that the P&L programmer would have come up with the solution to his problem so quickly if he had not had the toolbox code to use as a model.

Quite often, after several generations or iterations, a programmer will be able to develop generalized Macro code. A paper presented at this year's SUGI (see reference 8) presents a collection of Macros that solve table lookup problems through hashing techniques. This started from the development, use and enhancement of "prototype sample code."

**4. BUILDING AND MAINTAINING TOOLBOXES**

In order for toolboxes to be effective they must be supported and encouraged by all of the levels in the organization, from the junior programmers to the most senior managers. This commitment is crucial because without it the toolboxes will not get either the use or the exposure that they require. During the beginning phases of toolboxing it is very hard to get an organization to spend any effort on organizing a toolbox environment. In many cases, toolboxes will start from the bottom up. A single analyst or programmer will start a toolbox for his/her own use; it will then grow to be used by an entire project or system; and finally the organization as a whole will support it. At first, it appears that toolboxes will divert resources from the problem at hand. (One easy way to start a toolbox is to consult papers presented at SUGI, e.g., references 1-14). However, once an individual, organization or group commits itself to toolboxing, its merits become obvious and support for its maintenance and enhancement grows rapidly. This enthusiasm is good; but it must be controlled. The following should be addressed by any organization that plans on utilizing the toolbox concept:

- Coding Conventions
- Documentation Standards
- Code Walkthroughs
- Code Organization
- Maintenance and Control

Each of these is discussed in the following sections.

**4.1 CODING CONVENTIONS**

In order to be effective, toolbox routines whether they are generalized Macros or sample code, must be written clearly and effectively. Many organizations already have coding conventions. These should be rigorously enforced for toolbox code. Such code should be thoroughly commented to explain the purpose and function behind each step. The easier the code is to read and understand, the further its use will spread. Comments should give emphasis to a generic statement of the problem being solved.
For generalized Macro tools, coding conventions must include how the Macros and parameters are named and used. Some important conventions would include:

- Consistent parameter specification
- Use of the typical "SAS defaults"
- Meaningful Macro and parameter names

As is the case with all of the factors discussed in this section, it is important to have a "committee" which oversees and controls the toolboxes. This committee should review and screen all entries to the toolbox to ensure that the appropriate coding conventions have been followed.

4.2 DOCUMENTATION STANDARDS

It is well known that software without good documentation can not be used to its fullest. This is especially true of toolboxes. Each member (i.e., each Macro or piece of prototype code) must be thoroughly and completely documented. The documentation requirements are different for Macro toolboxes as opposed to prototype code toolboxes.

For Macro toolboxes, the documentation for each member should include:

- a clear and concise statement of the function of the Macro;
- complete descriptions of all inputs and outputs to and from the Macro;
- detailed explanations of each parameter, including what each parameter value (or class of values) does;
- sample executions of the Macro, including annotated output of both the SAS Log and Print pages;
- a complete list of any "side effects" of calling the Macro.

For toolboxes of prototype code the documentation should be heavily weighted towards a complete description, as generic as possible, of the problem. Almost as important is a detailed discussion of the techniques/code used to solve the problem along with the reason why the particular technique was chosen. Why a particular technique was used can be the most important. Consider the prototype specification that she merged in the new group requirements rather than using IF statements to create them because it made the solution data driven and thus easy to change.

4.3 CODE WALKTHROUGHS

Frequent walkthroughs should be held to discuss toolbox code and to maximize its exposure. The implementation of such walkthroughs is very different for generalized Macro tools as opposed to prototype code.

For Macro tools, walkthroughs should begin with a statement of the problem and should be geared towards collecting input on all of the required features. Quite often development will predicate this; however, it is still useful to have a walkthrough before the code is completed (and installed). In addition to this walkthrough, two other types of walkthroughs should be held. First, would be one attended by fellow programmers whose goal is to make sure that the code is clean and efficient. Second, would be one attended by potential users that addresses how the Macro is used and its potential "side effects".

For prototype code, walkthroughs should have more of an emphasis on the problem and a general discussion of how it was solved. In this way, the organization can maximize the exposure of the techniques. In many cases, the original author may not realize the full impact or utility of what they have developed. It is only through the input of others attending the walkthrough that prototype code be fully appreciated. Such walkthroughs may identify a future Macro tool that needs to be developed.

The committee responsible for toolboxes should make sure that these walkthroughs are scheduled on a regular basis and are convenient for staff to attend.

4.4 CODE ORGANIZATION

In order for any toolbox to be effective, the code must be centralized, available and accessible. It is recommended that the code be stored on-line on a specific userid/path/directory (location). This location should be devoted to toolboxes; it should not be shared with any other functions.

For Macro toolboxes, the member name should be the same as the Macro name. This has two advantages. It makes it easy to locate specific Macros and it also makes the use of the SAS AUTOCALL facility possible.

For prototype code toolboxes, an important criteria is that the member names, in addition to being unique, either be meaningful or if that is not possible, that a cross reference dictionary that maps the name into a brief description be supplied.

Such a cross reference is a good idea for both types of toolboxes. It should be located in a "member" of the toolbox library, perhaps called "AAHELP" (so that it comes out first alphabetically in, for example, PROC SOURCE).

4.5 MAINTENANCE AND CONTROL

As is the case with all code, toolbox code must be maintained. Since toolbox code will typically get wider exposure than application specific code, maintenance is particularly important. The importance of controlling the toolbox environment is also an issue of critical importance. In order for the code to be effectively used in developing applications, the general user must be assured that the toolbox contains code that works as described; the most up to date copy of the code; all of the applicable code (i.e., there are not an undefined number of toolboxes which the user has to first find and then review). Only if all of these conditions are met, can the use of the toolbox gain wide acceptance and use.

The committee that controls the toolbox environment should make sure that only designated staff have write access to the toolboxes. No unauthorized staff should be allowed to add something to or change a member in the toolbox. Unless the proper procedures are followed, the integrity of the toolbox is in jeopardy. It is important to make sure that only tools that meet the above conditions find their way into the toolbox library, with a designated member of the committee responsible for reviewing test runs to validate the accuracy and reliability of the code. It is also important for the committee to make sure that all useful tools make their way into the library. This requires that committee members come from a broad range of the organization and "beat the bushes" to make sure they are aware of what is going on in terms of SAS applications. If an organization permits a proliferation of toolboxes, most of the benefits are lost.

For Macro toolbox code, maintenance should begin when the code is installed in the toolbox. Before its installation the committee should validate that a complete test bed has been developed, run and its output saved for comparison with future test runs. All changes,
corrections and enhancements to the Macro should be coordinated through the committee which will make sure that the test bed is run again. If any new test cases are necessary, the staff member designated by the committee should add them to the test bed. The test bed should also be run whenever a new release of the SAS system is installed to make sure it does not introduce any problems.

For prototype code, there are two primary maintenance activities. First, the committee should validate that the code still works with new releases of the SAS System. Second, obsolete code should be deleted. Code can become obsolete when there are new and better ways of solving the same problem (i.e., new SAS products/features, Macros or newer prototype code).

5. CONCLUSION

All organizations can increase both their productivity and expertise by making sure that they widely distribute the techniques and methodology they have developed. This is true for both generalized and specific solutions to problems. You can never tell how someone else can use an idea or technique that you have developed.

Properly implemented, toolboxes facilitate communication, the spread of ideas, and prevention of duplication of effort.

The author can be contacted at:

Don Henderson
ORI, Inc.
Suite 1000
601 Indiana Avenue, N.W.
Washington, D.C. 20004
(202)337-2666

SAS is a registered trademark of SAS Institute, Inc.

6. REFERENCES


