

Program Verification Made Easier: Templates and Toggle Switches

M. Rita Thissen, Pharmaceutical Product Development, Inc., Morrisville, NC

Abstract

Program verification does not have to be tedious, even if the application is complex and undergoes recurrent modification. Many computing groups build libraries of validated SAS® programs and macros which can be adapted for use in new projects. Modifying these template programs may necessitate re-verification and documentation. To make the process easier, build verification code into the templates and control its execution through use of a "toggle-switch" macro variable. When the macro variable is "on", the code executes; when it is switched to "off", the code is commented out. Toggled code may include any of the following: LIBNAME definitions for test data or output files, program lines which generate extra output for an audit trail, data subsetting, additional information written to the LOG file or use of system options such as MPRINT or SOURCE2. This technique is useful for debugging during development and maintenance of the template programs as well as for generating verification documents at the time the program is completed.

Introduction

Guaranteeing the performance of software applications is one step in the adoption of good computing practices or quality management methods. The process is known as program verification or process validation. Regardless of the nomenclature, the goal is the same: to provide certification that the software performs according to specifications. Achieving this goal can be facilitated by designing software applications with toggle-switched code segments

which perform verification operations.

The Verification Process

Many programmers approach verification and other documentation tasks with reluctance, due to the perceived need for time-consuming and unrewarding effort between debugging and release. The way in which the verification process is handled can determine whether the work is in fact tedious or whether it is satisfying. Many verification techniques can be selected, with varying levels of effort and reliability.

Figure 1. Types of Verification Methods

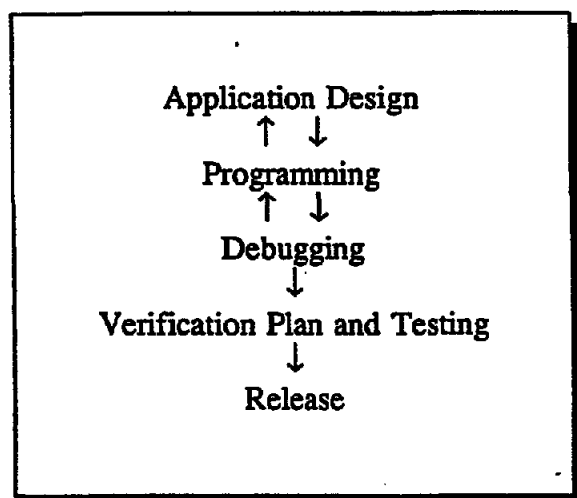
<u>Type</u>	<u>Comment</u>
Logical proof	Virtually impossible
Code review	As good as the reviewer
Alternate algorithm	Double work
Process reversal	Time consuming but convincing
Input to output checking for complete program	Very common, good for simple processes
Step wise input to output checking	Good for more complex processes

A common approach for validating an existing program involves generating a test plan and test

Applications Development

dataset, modifying the code to reference the test data and to provide additional output, and finally checking the output versus the input to confirm the results of the computation. The verification steps all take place between program testing and release (Figure 2). Programmers justifiably consider the planning, coding and checking as demands on their time, especially when the entire process must be repeated each time the application is updated.

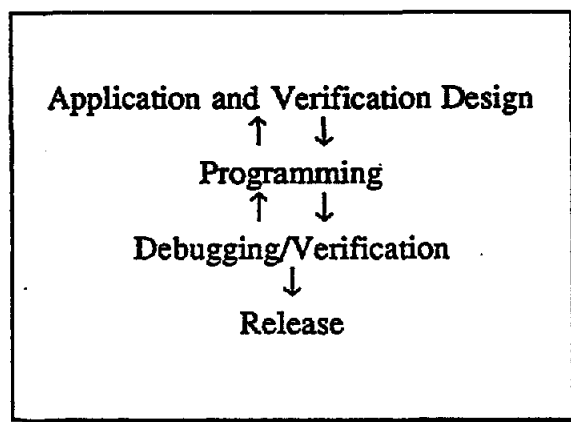
Figure 2. Application Development Process Components



To facilitate the programming process, design applications with built-in verification steps which can be switched "on" and "off" easily without removing the code from the program. Switch it on to generate validation reports as required for each new release, and switch it off to bypass those steps during routine use of the program. The "tedious" work can be minimized in this way: planning the verification takes place as part of planning the application; no modification of the code is required after debugging; and checking the output becomes an integral part of the debugging process (Figure 3).

This technique helps in maintenance of program libraries or collections of template programs because the test plan and verification programming tasks do not need to be repeated when applications are modified. Program libraries often contain two categories of applications: those which are used routinely without modification except for occasional maintenance, and those which act as generic templates to be adapted to each new task by changing portions of the programs themselves. Adding verification lines to the first category saves a modest amount of time because the need for re-verification is infrequent. Adding verification lines to the second category, the template programs, can save a greater amount of time because each application of the template should be validated separately. The extra effort spent during development is repaid many times over the lifetime of the software.

Figure 3. Alternative Application Development Process With Built-In Verification



Toggle Switches

It is easy to add verification lines to SAS programs in such a way that the code can be activated or deactivated at will, through the use of a macro variable that acts as a "toggle switch". Let this toggle switch macro variable

contain a null string or a blank during software development and testing, and let it contain an asterisk afterwards. By positioning the toggle variable at the start of each line of verification code, the statements can be "switched" from executable to comment statements when the toggle changes (Figure 4).

The value of the toggle variable can be assigned in several ways. As shown in Figure 4, it can be assigned within the program through a %LET statement. Other methods do not require modification of the program itself (Figure 5), although these require more advanced programming techniques.

Figure 4. Using a Macro Variable as a Toggle Switch

```
* Code switched on if toggle=blank;
* Code switched off if toggle=*;
%let toggle = *;
.
.
.
libname source "...";
&toggle proc print ... ;
&toggle title "Print intermediate
               results for verification";
.
.
.
```

Care must be taken when using a toggle in conjunction with macro program statements. Adding an asterisk at the beginning of a line of macro code does not have the same effect as adding one to a base SAS or PROC statement.

For example, an asterisk in front of a %LET statement has no effect on that statement, but it can turn the next one into a comment. When

using toggle variables with macro code, it is better to perform a deliberate check on the value, as shown in Figure 6.

Figure 5. Passing a Value to the Toggle Variable

- SYSPARM parameter passed to the program
- Conditional value assignment based on input data values
- Conditional value assignment based on error conditions

Templates and Toggles

Planning an application such as a template program with toggled verification code requires a little extra effort to include the necessary output statements, but it makes debugging simpler each time the template is applied. Ideally, the input data (or a representative subset) and the results of each data step should be reported in the verification output, so that a paper trail of all data manipulation is generated. Transformations through standard SAS procedures do not need to be verified per se, because the SAS software system itself has been validated before release, but adding PROC step output to the verification report can make the report easier to check. Add informative titles and footers to the output and extra information to the SAS log so that an independent reviewer can check the results easily. Use toggle variables on all these statements so that the output and SAS log files do not contain unnecessary information after release of the application. By building the verification code into the application design, and by employing toggle variables to control its execution, both debugging and verification are combined into a

Applications Development

single stage: the detailed output aids in debugging during development and the final, fully correct SAS log and listing serve as the process validation documents. For templates used repeatedly in similar applications, the ease of switching from development to release makes toggled verification code invaluable.

Figure 6. Use of Toggles With Macro Code
-- Beware!

```
%let toggle = *;
* This will not eliminate the
  %LET but will eliminate the
  PROC PRINT. Do not do this!!;
&toggle %let mvar = 3;
proc print ... ;
.
.
.
* This is what to do instead, within
  a macro program;
%if %str (&toggle) = %str ( )
  %then %let mvar = 3;
.
.
.
* If you are not writing macro
  code, do this;
data _null_;
  if "&toggle" NE "*"
    then call symput ('mvar', '3');
  else call symput ('mvar', '');
run;
.
.
.
* The next line does not work
  correctly if toggle resolves to a
  space. Better not risk it;
%&toggle.let mvar = 3;
```

Figure 7. Selected Examples of Statements
Which Might Be Toggled

<u>SAS Statement</u>	<u>Verification Use</u>
• OPTIONS SOURCE2 MPRINT SYMBOLGEN	• Add information to SAS log
• LIBNAME	• Reassign librefs for test data libraries
• FILE or FILENAME PROC PRINTTO	• Redirect verification output
• WHERE statement IF statement	• Subset or restrict data for testing and verification
• PROC PRINT PUT LIST	• Record additional information for verification
• PROC FORMAT FMTLIB	• Record format values for verification of their usage
• TITLE FOOTNOTE	• Add information to verification report to facilitate checking

Many SAS statements can be toggled to change an application from verification to release mode and back again (Figure 7). Librefs can be redefined during verification by overriding the usual definition with a test-specific library reference, allowing test input files to be kept in separate data libraries. Verification documents can be redirected through toggled FILE and PROC PRINTTO commands. System options and PUT statements can be toggled to provide extra information in the program LOG files.

Any information useful for debugging can be installed with a toggle switch to facilitate verification and maintenance without hampering routine usage.

Summary

Program verification processes certify that an application performs the job it is intended to do. The documents generated during verification provide a permanent account of the actions of the application on the data, showing evidence of good programming practices and quality data management. By designing applications with toggled verification code, the documents are easily generated, easily suppressed, and can play a useful role as development tools in addition to becoming permanent records.

Acknowledgements

The author wishes to thank PPD for supporting this work and the members of the Clinical Computing Department for their contributions in developing the approach and improving the manuscript.

For further information, contact the author at:

Pharmaceutical Product Development, Inc.
1400 Perimeter Park Drive, Suite 100
Morrisville, NC 27560
Phone (919) 380-2000
FAX (919) 380-2022
Internet thissenmr%ppdrtp@mcimail.com