Paper 95-27

A Version Control Kluge for SAS® Programs - using SAS!

Tim Williams, PRA International, Charlottesville, VA

ABSTRACT

Programming with the SAS Software® language presents technical challenges similar to those experienced in other languages such as Java, C++, or Visual Basic. This is especially true when a widely distributed programming team collaborates on code development. To ensure the success of a project, the team requires effective communication and the ability to implement a version control system to protect their work. Version control of SAS programs is mentioned in SAS online discussion groups at regular intervals, often with no resolution other than the purchase of expensive and proprietary software.

Intended for an intermediate to advanced audience, this paper describes the implementation of a version control system for SAS programs using the SAS language itself. The solution was developed on MS Windows 98® using advanced macro techniques, SAS's facility to send email, and integration with other software packages including Micro Soft Outlook®, UltraEdit® as a code editor, and the freeware version comparison utility ExamDiff.

Originally coded using version 6.12 SAS Software, the paper also explores the potential benefits and limitations of implementing the system other versions. Concepts in this paper will interest a wide range of audiences who use SAS Software.

Keywords : version control, code development, programming, project management, SAS Software Macro Language, UltraEdit

INTRODUCTION

Large and complex SAS software projects often require a programming team in order to meet tight timelines and balance workload. When several programmers work on a project there is a risk that previous work may be accidentally overwritten by another team member. Complexity of the situation increases when the team is distributed over a wide geographic range. Even when not working in a team, you may often wish you could step back to a previously archived version of your code to quickly determine what seemingly innocent change is causing a once elegant program to suddenly cease functioning. These situations call out for a simple and effective version control system.

Some key attributes of version control systems include:

- A file repository with the capability to:
  - Check-out files for editing
  - Check-in files for protected storage and archiving
  - Prevent team members from overwriting work by others
- Change history with the ability to step back to previous versions of code
- A file comparison facility to compare previous edits with current changes
- The ability to identify who is currently editing a program.

An example of a commercially available solution is Micro Soft's Visual Source Safe®. Open-source solutions include the Concurrent Versioning System (CVS) and the Revision Control System (RCS). These alternatives satisfy many users, but commercial products are cost-prohibitive and open-source software may be intimidating to inexperienced users. Using a SAS-based solution affords SAS programmers a high level of comfort with the underlying code and allows them to adapt the system for their own use.

This paper describes a rudimentary form of version control for SAS programs designed for use by a geographically separated programming team. Built-in email messaging serves as the primary means of communication among team members. The concepts discussed can easily be applied to less complex situations such as a single programmer who wishes to keep a program archive during code development.

BACKGROUND

PRA International is a Contract Research Organization (CRO) that provides programming services to pharmaceutical and biotechnology companies. The need for version control came about as the result of our programming team in Virginia assisting one of our sister offices in Europe. Timelines were tight and we needed an efficient way to coordinate code development on a series of complex programs and macros. Some files required regular updating as the project evolved so there was a definite risk of accidentally overwriting changes made by other team members. Interoffice communication was further complicated due to the difference in time zones between the two offices. Email is our primary means of rapid communication within our company, so it was a natural choice to integrate this functionality into the solution.

The following definitions will assist in understanding the concepts presented in this paper:

DEFINITIONS

Check-out	To move files from the file repository to an editing folder (directory). *Note: The terms folder and directory will be used synonymously throughout this paper.*
Check-in	To move files from an editing folder (work directory) back into the file repository.
Work Directory	The location on a local server where files are copied to when checked out from the file repository. Programmers on the team have their own independent work directories.
File	Source of files for programmers. You check

Repository files *out* of the repository to your work directory to edit them, then check the files back *in* to the repository when done.

SETUP AND ENVIRONMENT

NETWORK AND DIRECTORY STRUCTURES

A hypothetical network configuration illustrated in **Figure 1** is used in the following discussion. SERVER1 in Country "A" acts as the file repository for a project from a hypothetical company named "DrugCo." Three programmers are working on a project for DrugCo code named "Project1." We develop standard directory structures for all projects where client and project name form the root of the directory and SAS programs are located in subdirectories based on their type and function (see **Figure 1**, Server 1 directory structure).

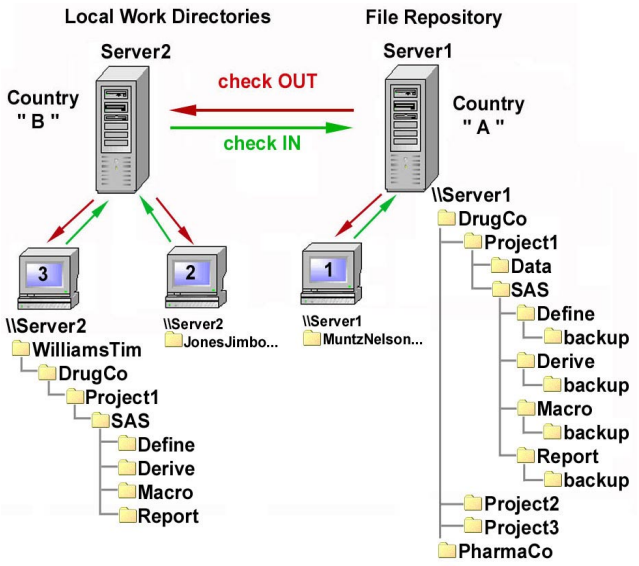

Figure 1 : Programming Environment.

Each SAS subdirectory in the repository must contain a folder named \backup where time-stamped copies are produced whenever a file is returned to the repository.

A single lead programmer maintains the version control process for the team, including initial setup of directory structures and the version control code. All programmers on the team must have identical work directory layouts at the "Client" level and below (see Server 2, workstation 3 for Tim Williams in **Figure 1**). As is the case for Programmer 1 (whom I shall designate as Nelson Muntz), the work directory can exist on the same server as the file repository, for example:

\\SERVER1\MuntzNelson\DrugCo\Project1\SAS\Define

Locating work directories on a local server cuts down on network traffic and allows you to work in an area that is isolated from both the other team members and the main file repository. Individuals check files out of the repository on SERVER1 and edit them in their own respective work directories. The files are checked back into the repository when editing is complete.

SETTING ENVIRONMENT VARIABLES

To operate the version control system, a program named `VersionControl.SAS` is submitted locally to PC-SAS running on your workstation. The program must be able to identify who is running the version control code so it can determine how to route messaging. For example, errors in file movement requests should only go to the person requesting the file, not to the entire programming team. This is accomplished by setting an environment variable in the `autoexec.bat` file:

```
SET USERID=<lastnameFirstname>
```

Example: `SET USERID=WilliamsTim`

The value of the `USERID` parameter must match exactly with the value assigned to the `myName<n>` macro variable assigned in the `VersionControl.SAS` program (discussed below). Following a consistent naming convention such as last name followed by first name is helpful.

TRANSACTION LOG FILE

An ASCII text file serves as a file transaction log, recording the name and path of a file being moved, its movement direction (*in* or *out* of the repository), the email address of the programmer initiating the file move, date and time of the activity and an optional comment describing the nature of the transaction. This file typically resides on the server housing the file repository. In the current example it would reside here:

\\SERVER1\DrugCo\Project1\SAS\report\fileStatus.txt

An excerpt from the transaction log file would appear similar to the following:

```
report\l_demog.sas ADD-NEW    williamsTim@praintl.com 02APR01 16:08
                              Listing of Demographic Data
report\l_demog.sas CHECK-OUT jonesJimbo@praintl.com  02APR01 16:53
                              Code Validation Checking
report\m_resp.sas  ADD-NEW    MuntzNelson@praintl.com 02APR01 17:01
                              Derivation of Response Data Set
report\l_demog.sas CHECK-IN   jonesJimbo@praintl.com  02APR01 17:45
                              Code Validation Complete:Program OK
```

The log file shows that the program l_demog.sas was written and added to the repository by Tim Williams, checked out and validated by Jimbo Jones, and passed the validation evaluation.

As a potential improvement to the log file, an auxiliary macro could be run at the end of day that first archives the transaction log, then sweeps the log's content to remove all files that were checked out, edited, and then returned to the repository. Only those files that are still checked out would remain in the updated transaction log. An optional email message could be sent to programmers who owe files to the repository, reminding them to check-in their code when finished editing.

CONFIGURING THE SAS PROGRAMS

Two SAS programs must be copied onto each programmer's local workstation for submission to a local PC-SAS session:

```
versionControl.sas
call_VC.sas
```

The following changes to the code are necessary, bearing in mind the details of the hypothetical situation discussed above.

VERSIONCONTROL.SAS

Macro variables in `VersionControl.SAS` define the location of the repository (***origDir***) and the location and name of the file that will act as the transaction log (***logFile***) :

```
/* Repository */
%let origDir= \\SERVER1\DrugCo\Project1\SAS;
/* File Transaction Log */
%let logFile=
\\SERVER1\DrugCo\Project1\SAS\report\
fileStatus.txt;
```

A list of parameters for each programmer working on the project appears in a subsequent section of code. The data includes their user name (***myName***<n>, used to match with the environment parameter `userID` set in autoexec.bat as described above in the section "Setting Environment Variables), email address (***email***<n>) and location of their personal work directory ***editDir***<n>. In my example, I would define the three programmers working on the DrugCo project using the following code:

```
/* 1. Tim Williams */
%let myName1=WilliamsTim;
%let email1=WilliamsTim@praintl.com;
%let
editDir1=\\SERVER2\WilliamsTim\Client1\Project1\sas;

/* 2. Jimbo Jones */
%let myName2=JonesJimbo;
%let email2=JonesJimbo@praintl.com;
%let
editDir2=\\SERVER2\JonesJimbo\DrugCo\Project1\sas;

/* 3. Nelson Muntz */
%let myName2=MuntzNelson;
%let email2=MuntzNelson@praintl.com;
%let
editDir2=\\SERVER1\MuntzNelson\DrugCo\Project1\sas;
```

The number of programmers active on the project is assigned to a macro variable named ***numStaff***. The value of ***numstaff*** is used in a macro `%DO` loop to send email notifications of check-in/out to the first ***&numStaff*** team members of a project listed in the code.

```
/* Number of programmers on the project  */
%let numStaff=3; /* 3 in my example*/
```

If ***numstaff*** is set to 3, then the first three staff listed in program will receive emails. This is a convenient way to add or remove team members as the project evolves.

Setup of `versionControl.SAS` may appear tedious, but it is only necessary at the start of a project, or when a new programmer is added and must have their parameters defined. Required setup is accomplished rapidly by designating a lead programmer to take care of these issues for all team members.

CALL_VC.SAS (CALLING THE MACRO)

Moving files in and out of the repository is as simple as using the following code which represents the entire `call_VC.sas` program:

```
%include "C:\data\proj\versionControl.sas";

%inOrOut( moveIt=IN,
          editFile=l_demog.sas,
          subdir=report,
          addlMsg=)
```

It is common practice to keep `call_VC.sas` open while working on SAS programs in other windows of the editor. The first line of the program designates where the version control SAS program is located on your workstation. The second line is a call to the macro with the following parameters:

Parameter	Value	Description
moveIt	ADD	Add a new file to the repository
	OUT	Check a file out of the repository
	IN	Check a file in to the repository
editFile		Name of the SAS program to move, including the .sas extension
subDir		Location of the file below the \SAS directory for a given project. In the example : \\SERVER2\DrugCo\Project1\SAS\report you would specify : `report`
addlMsg		Optional message text

USING THE SYSTEM

ADD A NEW FILE TO THE REPOSITORY

In the example below, the file `l_demog.sas` is checked in to the repository for the first time. If, for example, the programmer is Tim Williams, it is copied from his work directory here:

\\SERVER2\WilliamsTim\DrugCo\Project1\SAS\report\l_demog.sas

to the repository location on Server 1:

\\SERVER1\DrugCo\Project1\SAS\report\l_demog.sas

The parameters used to effect this transfer are:

```
%inOrOut( moveIt=ADD,
          editFile=l_demog.sas,
          subdir=report,
          addlMsg=Listing of Demographic Data
         )
```

As shown in **Figure 2**, the program first determines if a file named l_demog.sas already exists in the repository's ...\SAS\report subdirectory. If the file already exists, then an error message is emailed back to you stating that the file already exists and cannot be added to the repository.

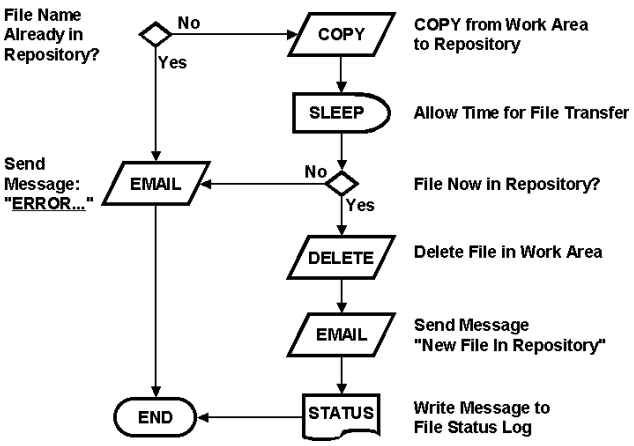

Figure 2. Flowchart of the Process for Adding a New File to the Repository

If the file is indeed new, it is then copied over to the repository. After a brief pause to allow time for the copy operation, the program checks to ensure that the file now exists in the repository. Unsuccessful copy operations result in an error message sent by email to the person who originated the request.

After a file is successfully copied to the repository, the original copy on your workstation is deleted and an email is sent to the programming team informing them that the file has been added and is now available to everyone. A line describing the successful addition of the file is also written to the transaction log file.

The advantage of using the version control system for file additions (compared to manually copying new files into the repository) is that all programmers are made aware of the existence of the new file.

CHECK-OUT A FILE FROM THE REPOSITORY

File check-out is accomplished using the following call to the macro:

```
%inOrOut( moveIt=OUT,
          editFile=l_demog.sas,
          subdir=report,
          addlMsg=Code Validation Checking)
```

In the check-out process (**Figure 3**), the version control program first checks to see if the file is available for check-out by determining if the file exists and that its file attribute is not set to read-only. If the file does not exist, a message is sent back to you stating that a file by that name does not exist in specified subdirectory of the repository. Similarly, if the file exists but its file attribute is set to read-only, then a message is sent to you stating that someone else has the file checked out for editing.

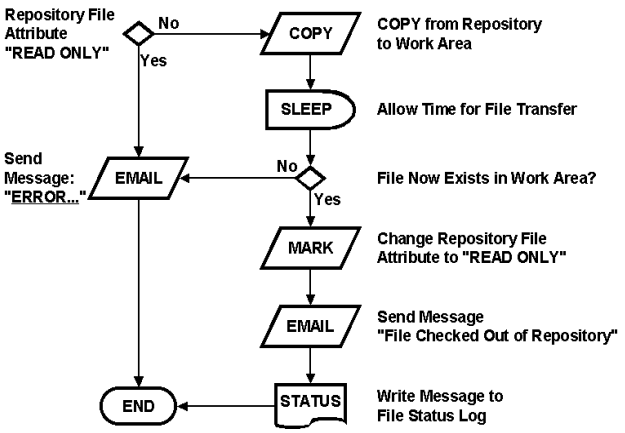

Figure 3. Flowchart of Check-out Process

When the file is available it is copied from the repository to the work directory specified in the macro call. If programmer Jimbo Jones initiates the transfer, the file is copied from the repository location here:

\\SERVER1\DrugCo\Project1\SAS\Define\report\l_demog.sas

to his work directory, here:

\\SERVER2\JonesJimbo\DrugCo\Project1\SAS\report\l_demog.sas

An email notifies you of the copy failure if the transfer does not succeed. If the copy succeeds, then the file attribute of the repository is set to “Read-Only” and an email message is sent to the programming team that the file has been checked out. A line describing the successful check-out is also written to the transaction log file. You may now edit the file located in your work directory.

CHECK A FILE BACK IN TO THE REPOSITORY

When you wish to move a file from your work directory back into the repository, you specify a call to the version control macro similar to the following:

```
%inOrOut(moveIt=IN,
         editFile=l_demog.sas,
         subdir=report,
         addlMsg= Code Validation Complete:
         Program OK)
```

The file check-in process is slightly more complicated (**Figure 4**) because it involves the creation of a time-stamped backup copy of the original code. When a check-in event is initiated the program first determines if the file attribute on the repository file is set to read-only. If it is not, it may mean that the version control protocol is not being followed by another programmer and the potential exists that a check-in would over-write someone else’s code. When this is the case an email is sent back to you, stating that the file on the repository does not appear to be checked out, so you cannot check it back in. The check-in process terminates and you must determine the source of the discrepancy before your file can be returned.

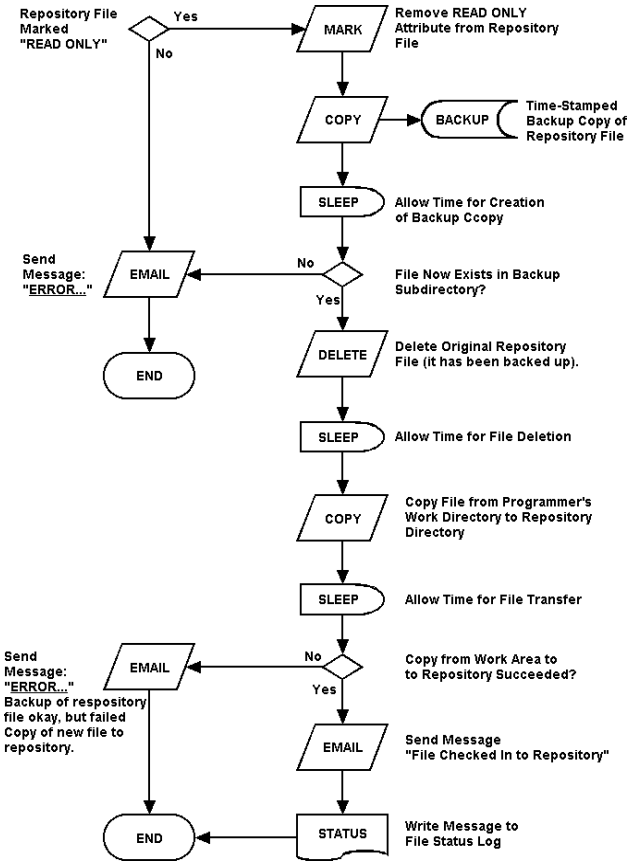

Figure 4. Flowchart of Check-In Process

If the repository file attribute is read-only, the version control system sets the attribute back to full access. The repository file is then copied to the backup directory and a time stamp is added to the file name as in the following example:

Original repository file: ...\SAS\Define\report\l_demog.sas
New backup copy: ...\SAS\Define\report\backup\
l_demog_02APR01_1745.sas

A new backup is created each time a file is checked in, allowing you to compare current code with a series of earlier versions. The backup filename contains both the date and time of the backup. In this example the file was created at 17:45 on April 2, 2001. If the program determines that the backup was not created, it emails you an error message and the check-in process is aborted.

After the backup is made the program deletes the original file in the repository and copies the file from your work directory to the repository. If this transfer fails, an email is sent to you stating that while a backup copy was successfully created in the repository, there was a failure when copying the work-area file to the repository. If the file is successfully copied from the work-area to the repository then an email is sent to the entire team stating that the file has been checked back in. A message is written to the file status log and the process terminates normally.

REVIEW OF FILE MOVEMENT PROCESS

You can easily move files to and from the repository by merely changing the values of four parameters. If the version control code **call_VC.sas** is kept open, you only need to change the ***moveIt*** and ***addlMsg*** parameters when moving the same file, so the time spent on the process is minimized.

```
%include"<DIRECTORY>\versionControl.sas";
%inOrOut( moveIt=<IN,OUT,ADD>,
          editFile=<filename.sas>,
          subdir=<directory name Eg:report>,
          addlMsg=<MESSAGE> )
```

CONFIGURING MICROSOFT OUTLOOK

An email message is sent whenever a file moves to or from the repository or when a transfer error occurs. This feature may generate large volumes of email so implementing mail-filtering rules is very important. Also, error messages are only sent to the person originating the file transfer and must be immediately visible to the programmer (and not deposited into sub-folder where they may be missed).

Filtering error message emails into your “IN box” and regular check-in/check-out mails to a designated “Project Folder” is easily accomplished using the filtering rules available in MS Outlook. Each programmer must set up a new folder within Outlook that is designated for the file movement emails. In my example the folder could be named DrugCoProj1.

Outlook’s rules wizard allows you to set up a rule to filter all successful file movement emails to the project folder within Outlook. The rule description would look similar to the following:

Apply this rule after the message arrives with DRUGCOPROJ1 FILE/ in the subject and with Move Status: successful transfer in the body
move it to the specified folder

All emails dealing with movement of the files to and from the repository have the project name in the subject line (**DRUGCOPROJ1** FILE/) and a message about the movement status (successful transfer or transfer error) in the email body - all generated automatically by SAS.

A message for a successful transfer may read as follows:

Subject : DRUGCOPROJ1 FILE/CHECK-IN : report\l_demog.sas
Email body text:
CHECK-IN
'File checked in the to the Repository OK'
File : l_demog.sas
Move Status : successful transfer
By : JonesJimbo@PRAIntl.com at 17:45 02APR01
Repository:
\\SERVER1\DRUGCO\Project1\SAS\report\l_demogsas
Edit Location:
\\SERVER2\Public\JonesJimbo\DrugCo\Project1\sas\report\l_demog.sas
Additional: Code Validation Complete. Program OK.

The email contains many details about the file being moved and the success or failure of the transfer operation.

ULTRAEDIT AS THE SAS CODE EDITOR

Our office currently uses the program UltraEdit to develop SAS programs as well as submit them to both local and remote instances of SAS. UltraEdit became popular as a code editor for

version 6 SAS while the programming community waited for a more robust editor within SAS software. UltraEdit is available for download at :

http://www.ultraedit.com/

The "Enhanced Editor" in Version 8 SAS software contains many features similar to those available in UltraEdit. Planned improvements to the Enhanced Editor will make the names and paths of files open in the editor available as environment variables. This will allow the version control process to launch directly from the editor, using code attached to the editor's toolbar buttons.

ADDITIONAL FEATURES : VERSION COMPARISON

Comparison of current and previous versions of SAS programs is made possible by using the time-stamped backup copies produced every time a file is returned to the repository. You may wish to quickly view changes to code made by other programmers, or simply review your own code history. While many text editors (including UltraEdit) allow file comparison, a freeware utility called ExamDiff is a very versatile and user-friendly solution. It is available for download at:

http://www.prestosoft.com/examdiff/examdiff.htm

LIMITATIONS AND FUTURE DIRECTIONS

The method described in this paper is only a partial solution to version control, hence my designation of it as a kluge.

kluge /klooj/
[from the German `klug', clever; poss. related to Polish `klucz' (a key, a hint, a main point)] 1. n. A Rube Goldberg (or Heath Robinson) device, whether in hardware or software. 2. n. A clever programming trick intended to solve a particular nasty case in an expedient, if not clear, manner

Readers are referred to the On-Line Hacker Jargon File for additional less-than-flattering definitions of this term that I sincerely hope do not apply here.

Since my program is only a partial solution, it is necessary to point out a few shortcomings:

- Lack of a robust file-locking facility. Changing the file attribute status to “Read Only” can be easily circumvented by another programmer in a Windows® environment.
- Windows®-specific commands are used for setting file access attributes. Code must be altered when working on other platforms.
- No integrated means of viewing the edit history of programs.
- Use of email as the primary means of communication between programmers does not work well in situations where some team members work off-line (without a continuous email connection).
- Lacks the ability to automatically open a file into the code editor when checking a file out of the repository.

The system would become much more automated if the program name and path were available as environment variables in the Enhanced Editor. We could then assign calls to the version control code to toolbar buttons and the user could easily click on the buttons to initiate file transfers. This is possible with editors such as UltraEdit, but the required parameters are not currently available in the Enhanced Editor of Version 8.2 SAS software. SAS technical support has communicated that the parameters are stored internally and not available to the programmer. Fortunately, they will be available in Version 9.

CONCLUSION

The method presented here is not intended to replace version control solutions required for large and complex projects. When using this alternative, a number of other supplementary tools are required to track projects. The version control program is no substitute for a well-organized team and good communication.

All SAS program code may be obtained by contacting the author. Space did not allow for a detailed discussion of the many SAS coding techniques used in producing this solution. Code excerpts will be presented at the SUGI27 conference.

REFERENCES

Bolinger, D and T. Bronson. 1995. Applying RCS and SCCS. O'Reilly & Associates, Inc. 501 p.

McConnell, S. 1996. Rapid Development: Taming Wild Software Schedules. Microsoft Press. 647 p.

The On-Line Hacker Jargon File. Version 4.2.3, 23 November 2000. http://www.tuxedo.org/~esr/jargon/index.html

ACKNOWLEDGMENTS

I wish to thank the Analysis Programming staff at PRA International for their feedback and assistance. Larry Broach, Bettina Grotz-Kettner and Dev Patel provided instrumental technical insights, suggestions, and testing.

CONTACT INFORMATION

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

Tim Williams
PRA International
4105 Lewis and Clarke Drive
Charlottesville, VA 22911
Work Phone: (434) 951-3504 (direct)
Fax: (434) 951-3001
Email: WilliamsTim@PRAIntl.com
Web: www.praintl.com