Paper 179-2014

Check It Out! Versioning in SAS® Enterprise Guide®

Joe Flynn, Casey Smith, Alex Song, SAS Institute Inc., Cary, NC

ABSTRACT

The life of a SAS® program can be broken down into sets of changes made over time. Programmers are generally focused on the future, but when things go wrong, a look into the past can be invaluable. Determining what changes were made, why they were made, and by whom can save both time and headaches. This paper discusses version control and the current options available to SAS® Enterprise Guide® users. It then highlights the upcoming Program History feature of SAS Enterprise Guide. This feature enables users to easily track changes made to SAS programs. Properly managing the life cycle of your SAS programs will enable you to develop with peace of mind.

INTRODUCTION

Tracking changes made to a program over time is very useful, particularly when collaborating with others. Sometimes, it is even a requirement for auditing purposes. One way this can be accomplished is by manually saving additional copies of programs as they change. Though effective, this is a very tedious and inefficient approach. We will explore more elegant solutions.

BACKGROUND

To employ version control in your day-to-day work in Enterprise Guide, you first need some basic knowledge of Enterprise Guide projects and how SAS programs are stored.

PROJECT FORMAT

An Enterprise Guide project file (.egp) is an archive in a proprietary format that stores SAS programs, SAS logs, ODS results, references to data, task states, and all the other information needed to reconstitute the state of an Enterprise Guide project. The project file often contains both embedded content and references to external content. Embedded content is stored wholly within a project file. For example, SAS logs and ODS results are usually embedded directly in a project file, whereas only references (or shortcuts) are stored within a project for externally referenced content; the actual content still exists outside the Enterprise Guide project. For example, *references* to data are stored in a project file, but the data itself lives in a data file or server. See **Figure 1**.

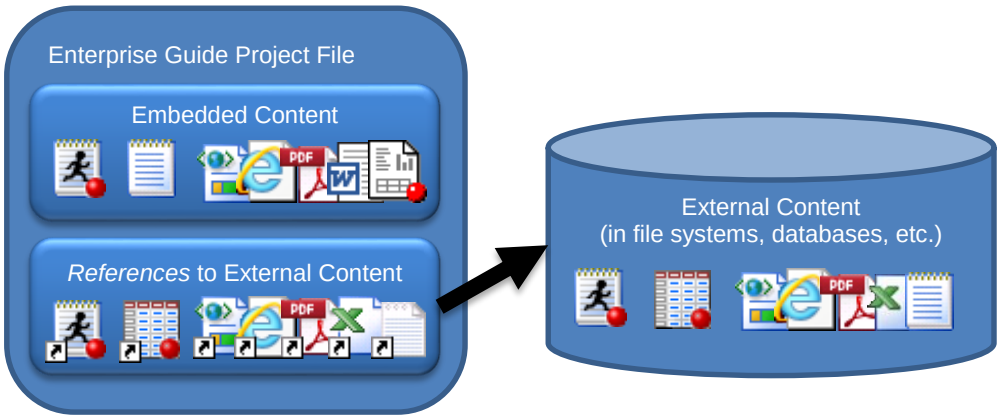

Figure 1: Embedded versus External Content in Enterprise Guide Projects

EMBEDDED VERSUS EXTERNAL PROGRAMS

Two different storage options exist when working with SAS programs in Enterprise Guide. Programs can be embedded in the project or externally referenced.

Embedded SAS programs are stored inside the Enterprise Guide project. When saving a project, imagine the SAS programs being added to an archive file for storage. Upon closing the project, the SAS programs can no longer be accessed directly. To access the SAS programs, the project must be opened (or unzipped). **Note:** When you create a new program in Enterprise Guide, it is initially embedded.

Alternatively, a SAS program can be stored externally to a location on disk (typically as a .sas file). In this scenario, the project stores the path to the SAS program as part of the project XML. No portion of the SAS code exists inside the project. All open and save operations are performed by directly reading from and writing to the SAS program on disk. When using this option, the program is available at all times, not only when the Enterprise Guide project is open.

There are a few ways to determine whether a program is embedded or referenced externally in a project. The first clue is in the Project Tree or in a Process Flow. When you examine the program icons, external programs have an arrow in the bottom left corner, as seen in **Figure 2**.

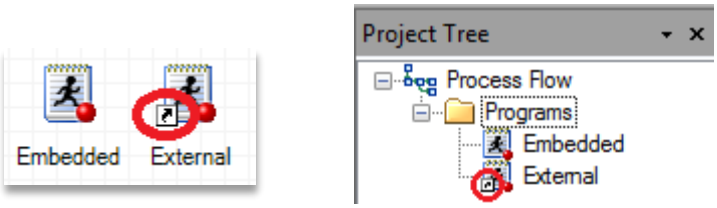

Figure 2: Shortcut Icon Indicating an Externally Referenced Program

Examining the program's properties is another way to determine how it is being stored. To access a program's properties, right-click a program icon and select **Properties**. By default, the General section is displayed, which contains a **File path** label. In **Figure 3**, the path is listed as **(Embedded In Project)**. As stated, this program is embedded in the project. The **Save As** button can be used to save the program externally. After doing so, the program in the project becomes a reference to the program file you saved externally. The program code is no longer stored in the project.

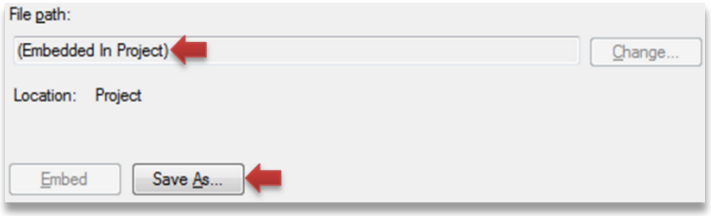

Figure 3: File Path of an Embedded Program

If the program is stored externally, the program's properties will display the File path location similar to **Figure 4**. Click **Embed** to embed the program code inside the project. The program is then an internal copy and no longer references the external file. The **Save As** button is also available to save a copy of the program to a new location and update the project reference.

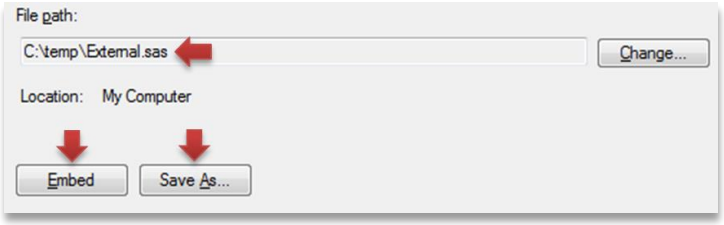

Figure 4: File Path of an External Program

Embedded programs cannot be easily managed in version control, because they are archived in Enterprise Guide projects. Enterprise Guide projects are binary files, which are inefficient to store inside version control systems. Also, the ability to see meaningful differences between versions is not available. This makes using version control with embedded SAS programs impractical. Using version control with externally referenced programs enables you to separate the source code from your project and is the recommended approach.

VERSION CONTROL

Version control provides the facility to manage and track changes to source files during the development process. Some SAS users manually save a new version of their program with each update. For example, you will find a directory with program1 to program32, all representing incremental changes made to the same program. Users manually save to prevent loss of work and to have a track record. It provides a safety net and enables them to make

modifications with the confidence that they can always roll back to the previous version. This process is an inefficient version control system. In the following example, we explore using a more formal version control system in conjunction with SAS Enterprise Guide.

VERSION CONTROL IN SAS ENTERPRISE GUIDE

Currently, there are no features implemented within Enterprise Guide that directly interact with version control systems. To use source control management with Enterprise Guide, save SAS programs that are external to the project and manage the source files independently. Many tools exist that are developed specifically for this purpose. We will walk you through an example using Git. However, the concept remains the same for any version control system.

SET UP DIRECTORY STRUCTURE

Begin by creating the folder structure in **Figure 5** under the root directory of your hard drive.

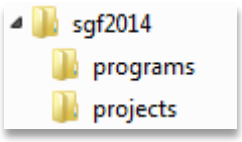

Figure 5: Folder Structure for Walk-through

OPEN THE TOPNCATEGORIES PROGRAM

The **TopNCategories** sample program, which is provided with all Enterprise Guide installations, is used as a template. To access this program in Enterprise Guide:

1. Select **File > Open > Program**.
2. In the browse dialog box, navigate to the Enterprise Guide sample code directory. By default, this is located under **C:\Program Files\SASHome\SASEnterpriseGuide\<version>\Sample\Code**.
3. Select the **TopNCategories.sas** program and click **Open**.

SAVE TOPNCATEGORIES INTO THE PROGRAMS DIRECTORY

After opening the program, it is displayed inside the editor. It is important to note that **File > Open > Program** was used to access the program. (Creating a new program by selecting **File > New > Program** would result in an initially embedded program.) This means that the project is externally referencing the **TopNCategories** program. This can be verified by clicking **Properties** on the editor toolbar, or by right-clicking the program in the process flow and selecting **Properties**. On the **General** page, the **File path** field indicates the full path to the SAS program. This is shown in **Figure 6**. Remember, if it were embedded it would say **(Embedded In Project)**.

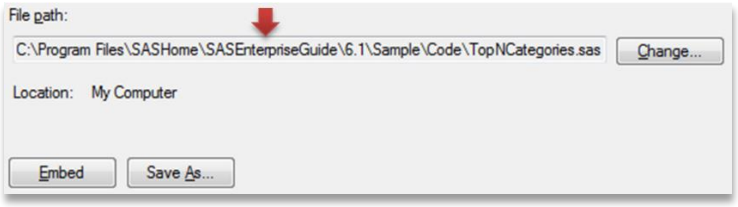

Figure 6: File Path of Externally Referenced Program

Save a copy of **TopNCategories** inside the **programs** directory created in the **Set Up Directory Structure** section. You can do this by clicking **Save As** inside the properties dialog box, or by closing the properties dialog box and selecting **File > Save TopNCategories as** and navigating to the proper location.

USE RELATIVE PATHS

Next, instruct Enterprise Guide to store the reference to this program as a relative path:

1. Select **File > Project Properties**.
2. Select **File References.**
3. Check the **Use paths relative to the project for programs and importable files** box, as seen in **Figure 7**.

Avoiding absolute path references grants you the freedom to store the project in any subdirectory, assuming the relative path from the project to the program remains the same. This ensures that the project remains flexible and is easy to share with coworkers.

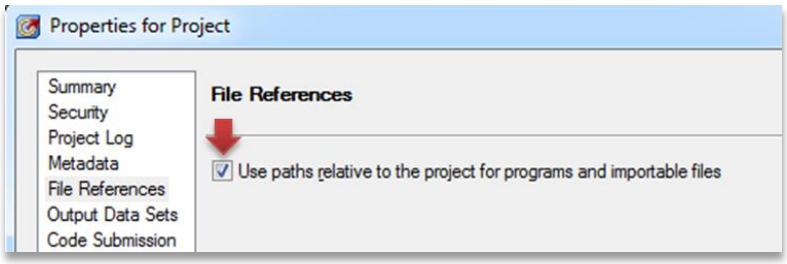

Figure 7: Option for Using Relative Paths for File References

SAVE PROJECT

Finally, save the project:

1. Select **File > Save Project**.
2. Navigate to the **projects** folder created in the **Set Up Directory Structure** section.
3. Enter the name **Top Categories**.
4. Click **Save**.

INITIALIZE GIT REPOSITORY

We are now ready to add version control to our externally referenced SAS program. To create a Git repository:

1. Install Git on your system if not already installed. (See http://git-scm.com/.)
2. Open the **Git Gui** utility.
3. Click **Create New Repository** on the main page, as shown in **Figure 8**.
4. Click **Browse**.
5. Navigate to the **programs** directory created in the **Set Up Directory Structure** section and select it as the location for the new repository. It should look similar to **Figure 9.**
6. Click **Create**.

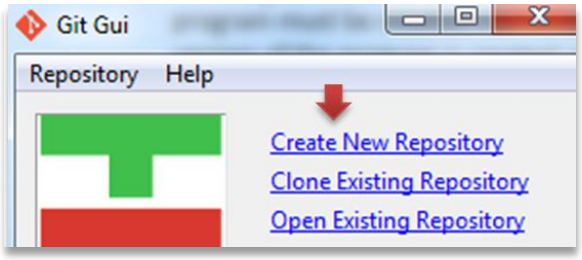

Figure 8: Create New Repository in Git Gui

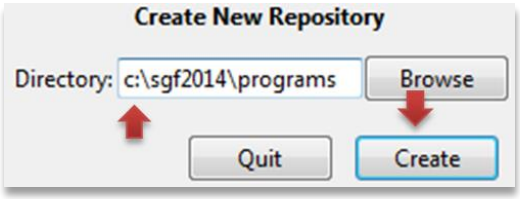

Figure 9: Specify Repository Location

COMMIT INITIAL VERSON OF TOPNCATEGORIES

When you create the repository, you will notice **TopNCategories.sas** is located in the Unstaged Changes area. We are happy with the state of the program and we want to save this version into the history. Complete these steps, which are also labeled in **Figure 10**:

1. Select **TopNCategories.sas** and review the changes.
2. Click **Stage Changed**.
3. Provide a message describing the changes.
4. Click **Commit**.

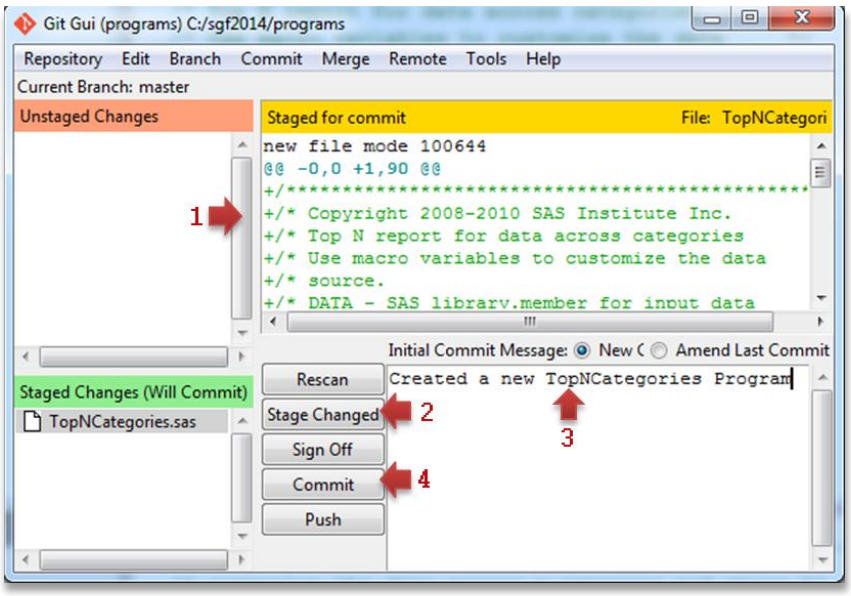

Figure 10: Initial Commit of TopNCategories Program

This version of the file has now been committed to Git and is part of the history. Select **Repository > Visualize masters history** to view the current history. The current history consists of a single commit of **TopNCategories.sas**, as seen in **Figure 11**.

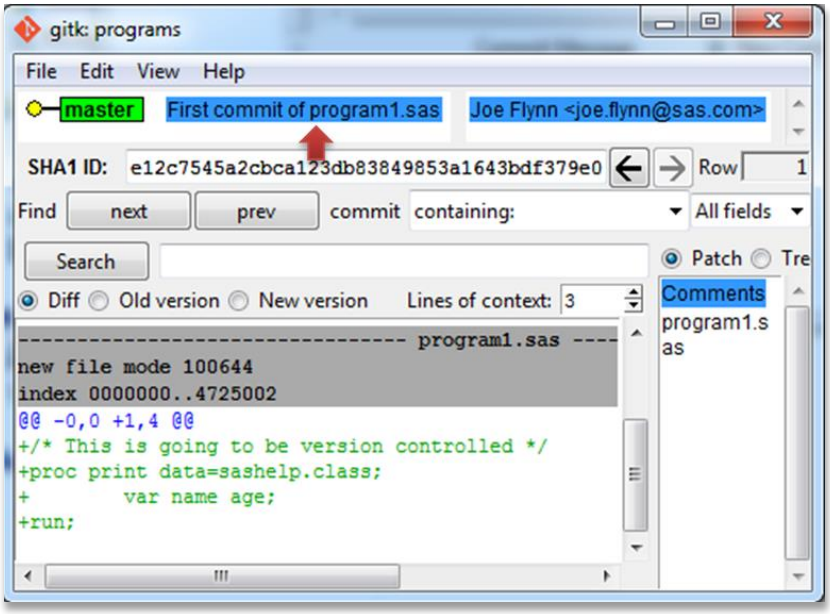

Figure 11: Viewing Repository History

MAKE CHANGES TO TOPNCATEGORIES

Now that we have a good first pass of the **TopNCategories** program in version control, we want to refactor the code a bit. Back in Enterprise Guide, open the **Top Categories** project and access the **TopNCategories** program. Suppose we want to refactor the first 'data' macro variable into three separate macro variables. We decide we would

rather have separate macros for the libref and data set, but still keep the data macro that is referenced throughout the program. To do this, we replace the following line of code:

```
%let data=SASHELP.CARS;
```

We replace that single line of code with the following three lines of code:

```
%let libref=SASHELP;
%let dataset=CARS;
%let data=&libref.&dataset;
```

Without rerunning the program (we are very confident in our changes), save your program by selecting **File > Save TopNCategories** and close the project. Saving the program writes the changes to the **TopNCategories** program on disk so that they can be committed.

COMMIT CHANGES TO TOPNCATEGORIES

To commit these changes to the Git repository, begin by opening the **Git Gui** utility. On the main page, you now see an Open Recent Repositories area. Click the link displaying the path to the repository created in the Initialize Git Repository section, as shown in **Figure 12**. This opens the repository.

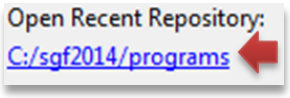

Figure 12: Open Recent Repository

Once in the repository you should see **TopNCategories.sas** in the Unstaged Changes area. When you select the program, you will see the file contents displayed as one red line and three green lines, as shown in **Figure 13**.

```
-%let data=SASHELP.CARS;
+%let libref=SASHELP;
+%let dataset=CARS;
+%let data=&libref.&dataset;
```

Figure 13: Changes Made to TopNCategories

This shows how the program has changed since the last time a version was committed. Everything looks as expected, so we can go ahead and commit the changes.

1. Select **TopNCategories.sas** in the Unstaged Changes area and click **Stage Changed**.
2. Review changes to the program in the Staged for commit area.
3. Provide a meaningful commit message detailing what was accomplished. For example, you can enter "Refactored data macro variable to reference new libref and data set macro variables."
4. Click **Commit** to finalize these changes and add them to the repository history.

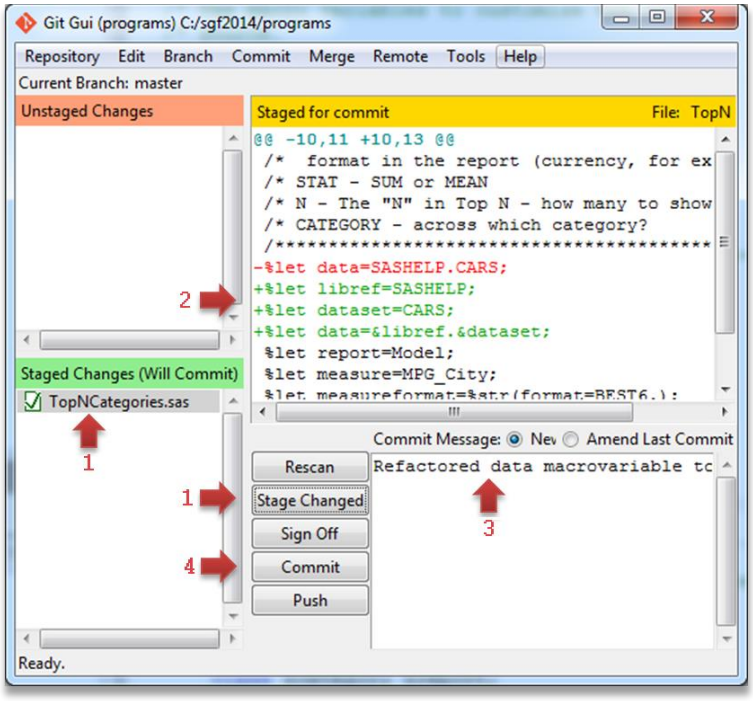

Figure 14: Committing Changes to TopNCategories

REVIEWING HISTORY

Suppose some amount of time passes and you have not revisited the **TopNCategories** program. To your surprise when running the program, 17 errors are returned in the log. You are certain it was functioning properly when the initial version was committed, but you do not recall exactly what was changed. To view the history of the program:

1. Open **Git Gui**.
2. Select the **programs** repository from the recent repositories list.
3. Select **Repository > Visualize Master's History**.

There are two commits listed, so it can be determined the only change made since the program was functioning must be part of the "Refactored data macro variable" commit. When you select the commit, the changes show exactly how the program was modified between commits, as seen in **Figure 15**.

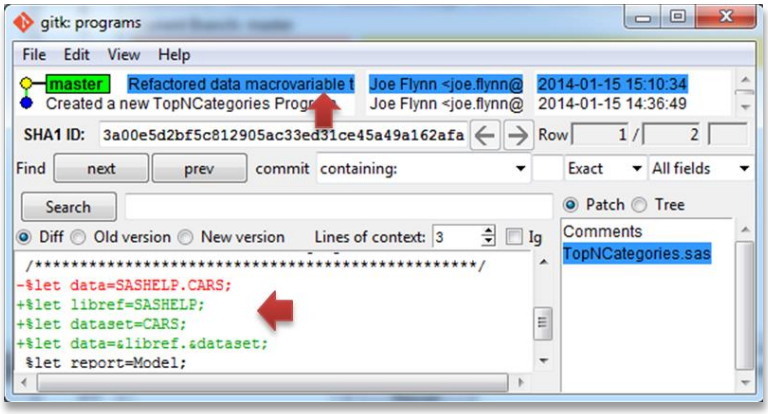

Figure 15: Viewing Repository History

To recap using source management control in Enterprise Guide today:

1. Store your SAS programs as (.sas) files.
2. Reference them in your Enterprise Guide projects.
3. Develop them using Enterprise Guide.
4. Track changes to them in an external version control system.

Although the example was extremely simple, version control is vital when working with complex source code. It helps determine when and why a problem was introduced into the code. Knowing the intent of the changes made to the program can save time. Sometimes changes are made to fix one bug but unintentionally introduce another. With this knowledge, you can target a fix for both issues.

PROGRAM HISTORY FEATURE

Beginning in the next release of Enterprise Guide (post 6.1), a new feature is being implemented to assist users with tracking changes made to embedded SAS programs. This program history will be embedded in the project and will provide invaluable debugging information to programmers. This feature will work out of the box, with no additional configuration steps required.

A distinction must be made between the internal program history feature and actual version control. Generally, version control has a centralized copy of all of the source and is frequently backed up. Unlike version control, the Program History feature is entirely embedded in the project and no centralized backup exists. If the project is lost, the history is lost along with it. This makes the importance of backing up projects even more vital.

The Program History feature will also provide some integration with external SAS programs that are under Git version control. Simple actions such as accessing a specific program's history, diffing versions, and committing new changes are all available from within Enterprise Guide. More in-depth operations such as push/pull/merge/branching/cherry-pick are not currently available and must be done outside of Enterprise Guide.

Note: Because the Program History feature and related functionality covered in the rest of this paper is still in development, the final implementation is subject to change. Although these features are not yet available, we will demonstrate how they are expected to work.

GETTING STARTED WITH PROGRAM HISTORY

Here, we point out where you can find the new Program History feature in Enterprise Guide. First, we create a new project by selecting **File > New > Project**. Name this project **Top Categories**. Next, we work with the sample program **TopNCategories** again. To add this program to your project, select **File > Open > Program** and navigate to your installed Enterprise Guide directory.

By default, it is located under **C:\Program Files\SAS\EnterpriseGuide\<version>\Sample\Code\TopNCategories.sas**.

Next, we need to embed this program into the Top Categories project. Right-click **TopNCategories** from the Project Tree and select **Properties**. Under the General section, click **Embed**. When a program is embedded in the project, you can take full advantage of the program history functionality. To learn more about the feature differences between an external and an embedded program, see the **External files** section later in this paper.

Save the project by selecting **File > Save Project Top Categories**.

PROGRAM ACTION TOOLBAR

Once you have the **TopNCategories** program open, look at the set of actions available immediately above the program's SAS code window. It should look like **Figure 16**.

Notice that there are three new action buttons located just before the **Properties** button:

- Changes
- Commit
- History

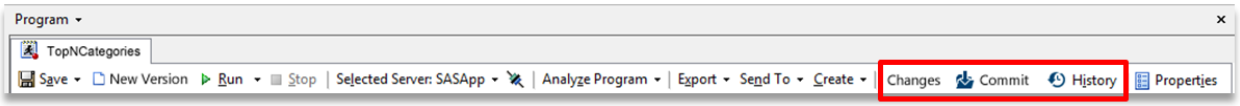

Figure 16: Program Toolbar with New Changes, Commit, and History Action

PREVIEW CHANGES

The **Changes** button shows you a quick view of the changes in your program since your last committed version. This lets you easily see what has been added or removed most recently.

To demonstrate this feature, make the following changes to your **TopNCategories** program:

Replace the following line of code:

```
%let data=SASHELP.CARS;
```

We replace that single line of code with the following three lines of code:

```
%let libref=SASHELP;
%let dataset=CARS;
%let data=&libref.&dataset;
```

After finishing the modifications, click **Changes**. A dialog box appears similar to the one in **Figure 17**. This dialog box highlights any changes made to the program since the last commit. Additions are highlighted in green, and removed code is highlighted in red.

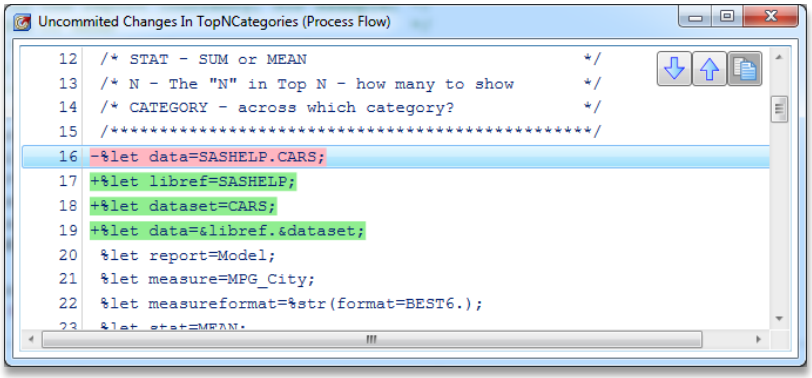

Figure 17: Changes Dialog Box

This is a quick and easy way to see what has changed since your last commit. If you want more robust functionality, you can install a third-party file comparison tool for features such as side-by-side comparison. For more information, see the **File Comparison Options** section.

COMMIT CHANGES

The next new feature on the program toolbar is the **Commit** button. If you have used Git as a source control provider in the past, you might be familiar with this functionality. The commit action is essentially how you create new versions of your program. As a best practice, commit your programs' changes at meaningful work boundaries.

Committing a Single Program

We just made some changes to **TopNCategories**, and we want to commit the changes. Click **Commit** from the program toolbar. You should see a commit dialog box, as seen in **Figure 18**.

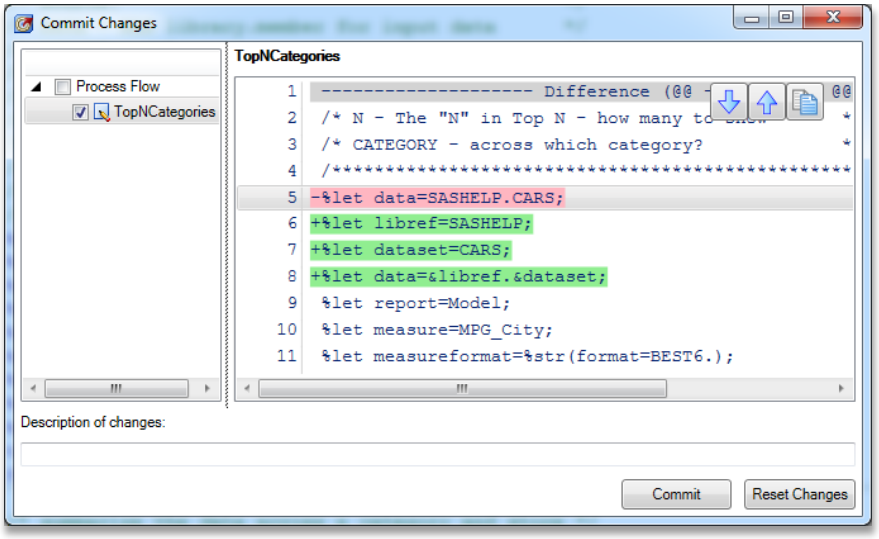

Figure 18: Commit Dialog Box

In the commit dialog box, you can see a preview of the changes to **TopNCategories** and a field for adding a description to describe your changes. It is required that you add a brief description of your changes for each commit. This makes it easier to find a previous version if you need to revert or look up changes. It also gives context to what was accomplished by the changes.

Committing Multiple Programs at Once

Over the course of your project, you might have multiple programs that are being modified at once. In this case, you will want to be able to commit multiple programs in your project at one time. When you have several modified programs, they will all appear in a list, grouped by process flow, when you click **Commit**. It should look like **Figure 19**.

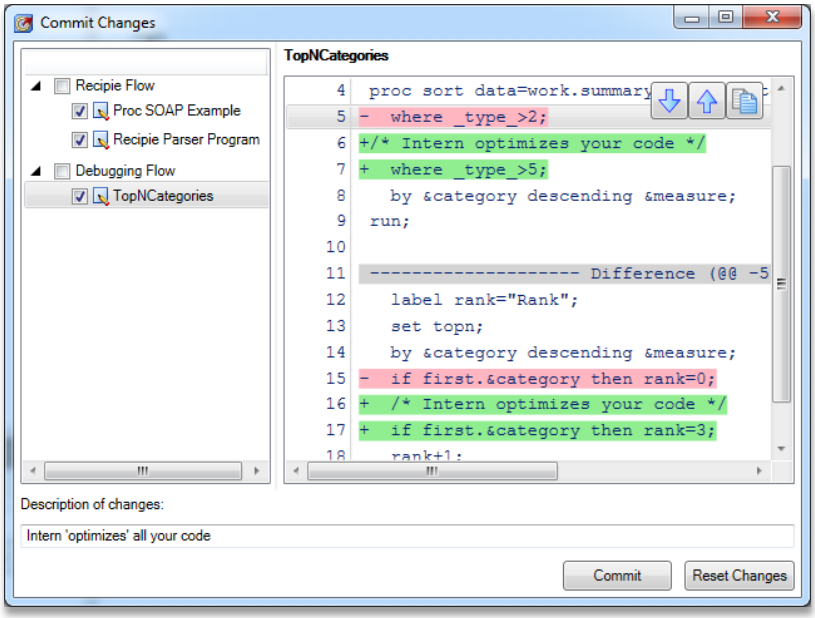

Figure 19: Committing Multiple Changes Simultaneously

Use the check boxes to select the programs you would like to include in the commit. It is a good idea to commit programs together when they are all part of a related change. For example, if you were to rename a macro variable in four different programs, you could commit them all at one time with a commit message similar to “Renamed the data macro variable to XXX.” A preview of each program’s changes is available by selecting it in the list view. A

description to label your commit applies to all programs selected at that time.

VIEW PROGRAM HISTORY

Once you have committed a version of your program, you might want to see the history of changes to your program. Click **History** to open a program history dialog box, similar to the one in **Figure 20**.

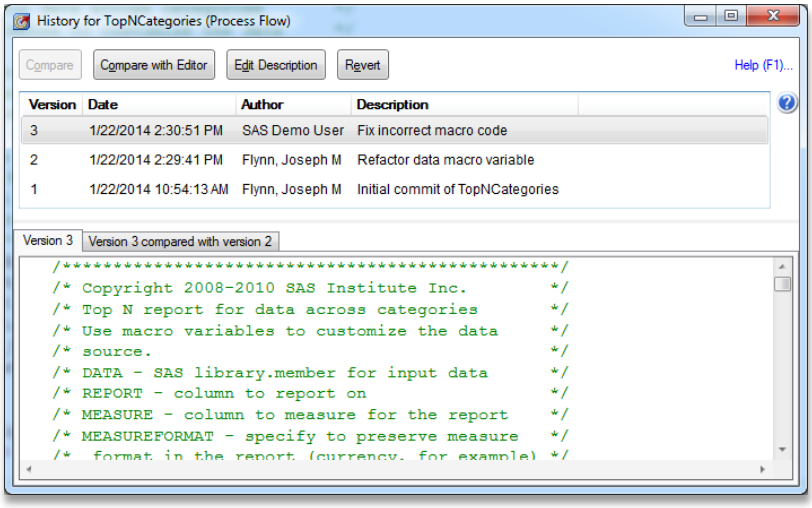

Figure 20: Program History Dialog Box

From your program history, you can see a list of committed versions of your program. You will see an assigned version number, the date on which the version was created, the author of the committed changes, and the added description for the version.

In the lower half of the dialog box, you will see two tabs. The first tab contains the full program as it was at that point in history. The second tab provides the differences comparing the selected version to its previous version, as seen in **Figure 21**.

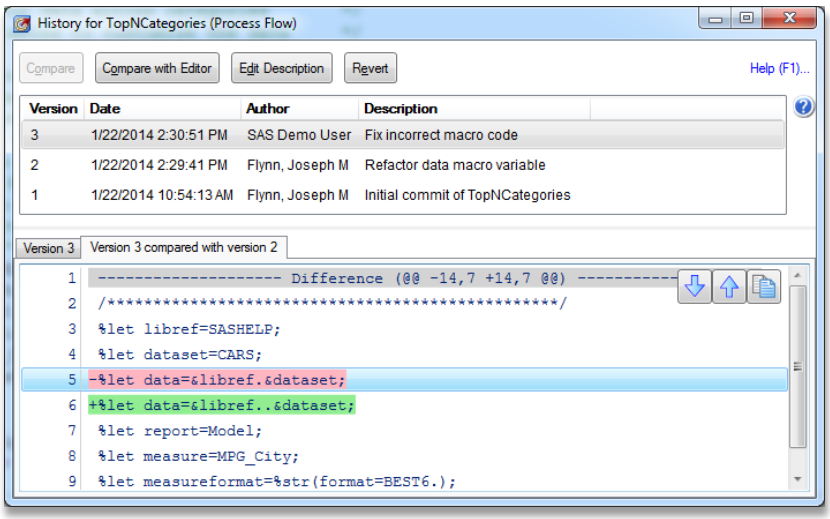

Figure 21: Default Comparison View

The following actions are available from the top of the program history dialog box:

- Compare
- Compare with Editor
- Edit Description
- Revert

Compare

When you select two versions (hold down the Ctrl key and select the versions you want to compare) from your program history list and click **Compare**, by default, you get a preview of the changes between the selected versions using the Enterprise Guide file comparison tool. The preview highlights added code as green and removed code as red, similar to clicking **Compare** from the program toolbar. In **Figure 22,** the first and third versions of the program have been selected. This highlights the changes between Version 1 and Version 3, which include the changes made in Version 2.

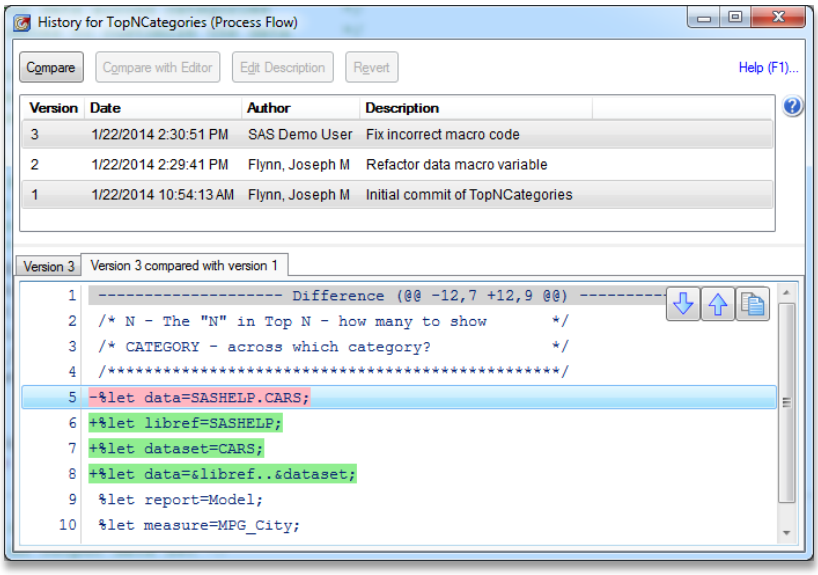

Figure 22: Comparing Selected Versions of a Program

If you have a third-party file comparison tool installed, clicking **Compare** launches the application and provides more robust comparison options, as seen in **Figure 23**. For more information, see the **File Comparison Options** section.

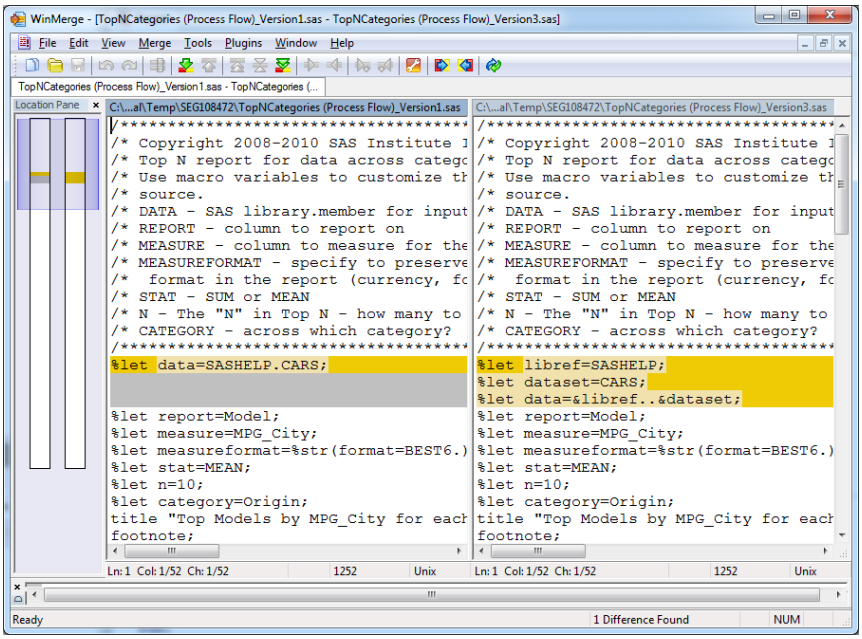

Figure 23: Example Results from a Third-Party Comparison Tool

Compare with Editor

The **Compare with Editor** action enables you to compare the selected version with the program that you are

currently working on.

Note: This is not necessarily the same as comparing the selected version with the most recent version because the program you are currently working on might not yet be committed. This enables you to compare your work-in-progress with a committed version.

Another Note: If you select the most recent version from the list and click **Compare with Editor**, this is the same as clicking **Compare** from the program toolbar!

Edit Description

When a version is selected from the list, clicking **Edit Description** enables you to edit the description field. As a best practice, provide a description of the changes you made to your program. This enables you to easily locate an older version later on. It also gives context for why you made the changes in the first place.

Revert

If, at any point, you need to go back to an earlier version, you can select an older version from the program history and click **Revert**. This action replaces your current program in the Enterprise Guide Program Editor with the selected version.

When you click **Revert**, a warning message appears, asking if you are sure that you want to replace your current program with the selected version. You will lose any unsaved and uncommitted work**.**

Collaborating on Changes

The program history keeps track of versions created by different Enterprise Guide users. In our **TopNCategories** example, you need some help with getting your project to run successfully. You decide to send your project to Sam, the residential expert on SAS programming. Sam opens your Top Categories project, opens the program history, and sees the changes you have made so far. He notices that there is an extra period missing from the data variable assignment.

He replaces the following line of code:

```
%let data=&libref.&dataset;
```

Here is the new line of code:

```
%let data=&libref..&dataset;
```

He commits the changes and sends the project back to you. You look at his changes in the program history and see that Sam's name now appears as the author of the latest version of **TopNCategories**.

Note: The author's name is derived from metadata. If you are connected to the server using shared credentials, this name might not be as meaningful.

COMMIT HISTORY

Earlier, we mentioned how you can commit multiple programs at once. When you do this, you might want to go back and see what files were committed at which point. The program history gives you the past versions for a single program only. To view a history of your commits for the current project, select **Program > Commit History**.

From the Commit History dialog box, you see a list at the top of the dialog box similar to the version list in the program history. This is a list of your commit actions for this project. The most recent commits appear at the top of the list. If you select a commit from the list, you will see a list of committed programs at the bottom of the dialog box. From the list of programs, you can select one and see the program code on the **File Contents** tab and a quick view of the changes that were made on the **Changes** tab. See **Figure 24**.

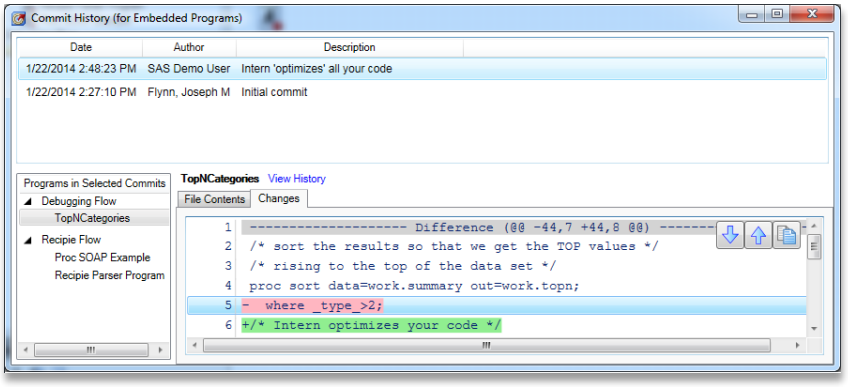

Figure 24: Commit History Dialog Box

From the project's commit history, you can also quickly navigate to the program history for the selected file. Click **View History** next to the name of the program above the **File Contents** and **Changes** tabs. You can also double-click on the programs listed under the process flow or right-click on a program and select **View History**. In addition, you can double-click on a version from the program history to open the commit history and see other programs that were committed at the same time.

CONFIGURING

The program history and file comparison tools can be configured by navigating to **Tools > Options**. There are two new sections: File Comparison and Program History. In File Comparison, we can specify a third-party file comparison tool (such as WinMerge) that handles side-by-side comparisons between versions. In Program History, we can manually or automatically commit new versions and clear the program history.

File Comparison Options

By default, Enterprise Guide offers a very basic view of your program's changes when you click **Changes** on your program toolbar. You can also select up to two versions of a program in the history dialog box and see a basic highlight of the changes.

Enterprise Guide also supports any number of third-party file comparison tools for a richer experience, including side-by-side comparisons and more robust navigational controls. You can use any third-party tool you want, but here are a few that you can try:

- WinMerge
- KDiff 3
- Beyond Compare 3

Specifying a Path to the File Comparison Tool

Once you have installed your third-party file comparison tool of choice, you need to tell Enterprise Guide which file comparison tool you are using. Navigate to **Tools > Options > File Comparison** and select your installed comparison tool from the drop-down list (**Figure 25**). If you have installed one of the tools from the list, the path to the tool is already filled in for you. You need to specify the path yourself if you installed the comparison tool in another location on your computer or if you installed another third-party tool that is not listed.

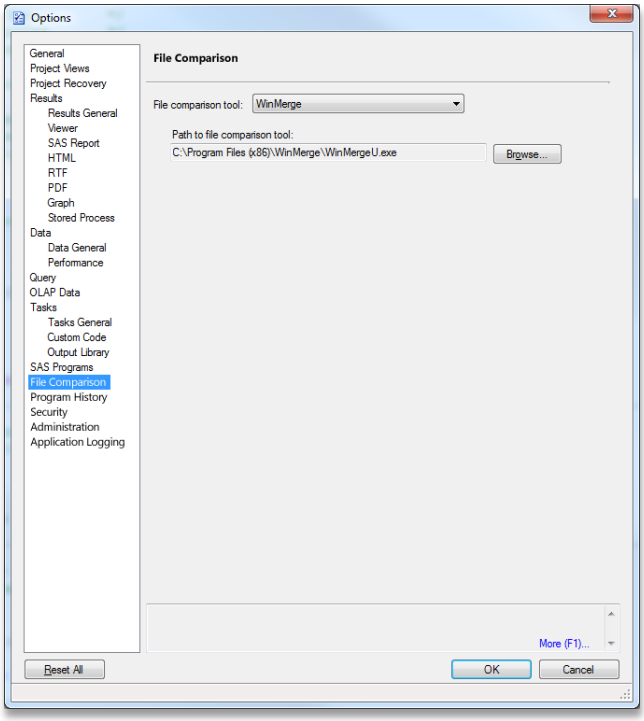

Figure 25: File Comparison Options

Program History Options

By default, the new Program History feature is enabled in Enterprise Guide so that you can manually commit changes to your programs. If you select **Tools > Options > Program History** (**Figure 26**), you can perform these tasks:

- enable or disable the program history
- choose to manually or automatically commit changes to your programs
- clear your entire program history

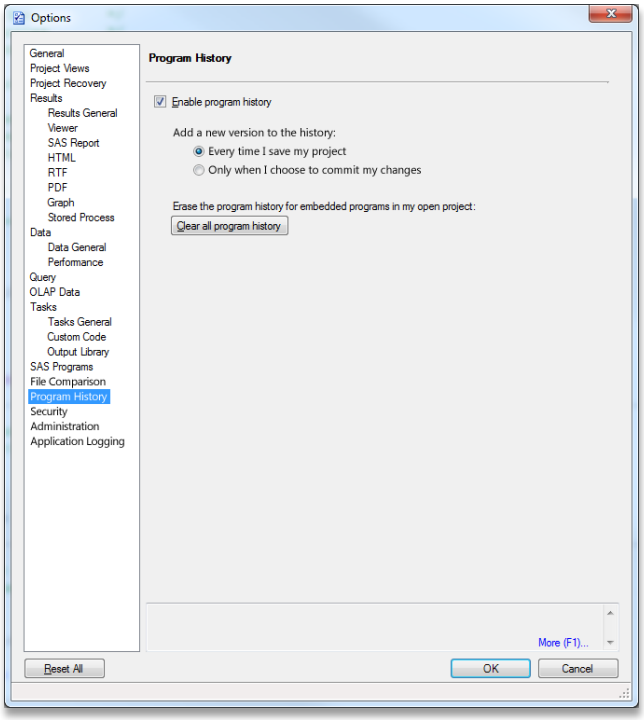

Figure 26: Program History Options

Enabling or Disabling the Program History

From the Program History options, deselecting **Enable program history** turns off the commit and program history buttons. Your saved history and commits are not erased, so you can enable the program history and continue viewing the history and making commits to your programs.

Note: The Program History feature does not consume resources unless used. Therefore, there is almost no benefit to disabling. However, we will likely provide the ability to disable the feature so that you can hide the related menu items from the user interface if you have no intentions of using the feature.

Manually or Automatically Committing Changes

By default, Enterprise Guide only commits changes to your program when you explicitly click **Commit** from the program's action toolbar. This enables you to manage exactly which programs in the project are tracked. It also gives you the freedom to decide when a new version is created. On the downside, this takes an extra action to accomplish. The Program History feature can be configured to commit all embedded programs that have changed each time the project is saved. This results in more frequent commits. However, this feature is helpful if you do not want to separately remember to commit a new version of your program.

Note: If creating a new version on save, keep in mind that the history grows very quickly if you save often. Also be mindful that all programs that have changed in your project will be committed. Certain programs (for example, extremely large programs that contain data lines) are not ideal for being versioned. Although just the differences are stored, if data lines are completely changed between saves, these differences can be quite large. This causes the size of the project to grow very quickly. Also, the save points might not be as meaningful if you are just saving to "not lose work." It is recommended to commit changes at meaningful points in your development cycle.

To change to automatically commit your programs, select **Tools > Options > Program History**. Under **Add a new version to the history**, select **Every time I save my project**.

Clearing All History

Over time, you might have an overabundance of versions collected in your program history. Typically, these versions do not take up a large amount of space, because only the differences between versions are saved (instead of making a duplicate copy of the file). However, it might become difficult to find the right version of your program if there is an extremely long list of versions.

If you would like to start with a blank slate, you have the option of erasing your entire program history. Note that this clears the history for all programs in the project. To preserve a historically accurate record of changes, Enterprise Guide does not allow you to individually erase versions of a program. Clearing history erases *all* previous versions from *all* programs in your project. This operation is applicable only to embedded programs and does not affect external programs.

To erase your program history, select **Tools > Options > Program History** and click **Clear all program history**.

EXTERNAL FILES

Simple integration with Git is planned for the next release of SAS Enterprise Guide. Many of the features discussed earlier for embedded programs will be available to external programs residing in Git repositories. However, here are some differences in functionality worth highlighting that apply to external files:

Program History

- Editing the description field is not supported. This field will always display the Git commit message.
- The Version column will not exist. Internal history is guaranteed to be linear. However, this is not necessarily the case with Git history.
- The current branch will be displayed.

Commit History

- Commit history will not be supported with external programs. Viewing the repository history must be done with an external tool.

ADDITIONAL COMPARISON FEATURES

We plan to complement the Program History feature with a couple of additional handy comparison features. You will be able to compare any two selected programs and preview unsaved changes to any program.

Note: These additional program comparison features will be available regardless of whether the Program History feature is being used. Both additional features will work with embedded and externally referenced programs. However, both require the use of a third-party file comparison tool. The third-party file comparison tool must first be configured in the Enterprise Guide options. For instructions on how to configure, see the **File Comparison Options** section.

COMPARING SELECTED PROGRAMS

Someone smart once said, “Imitation is the sincerest form of flattery.” (BrainyQuote.com[1] says it was Charles Caleb Colton.) Based on that quote, one could infer programmers are notorious for flattering, because we copy code constantly and often our own! Sometimes as a starting point, other times as a real-world approach to get the job done, it is quite common to maintain similar programs modified for slightly different purposes.

The Program History feature will enable you to compare any two versions of the *same* program. We are taking that a step further by enabling you to compare the current version of any two *separate* programs within your Enterprise Guide project.

This feature will be a convenient way for managing changes to similar programs within an Enterprise Guide project. You will easily be able to compare programs to see how they are different or have diverged from one another.

To use this feature:

1. In the Project Tree or in a Process Flow (**Figure 27**), select two programs. (Hold down the Ctrl key while clicking the programs to select both.)
2. Right-click on either of the programs to display the pop-up menu.
3. Click **Compare**.

The current version of the two selected programs will then be opened into and compared by your third-party file comparison tool. For an example, see **Figure 23**.

[1] http://www.brainyquote.com/quotes/quotes/c/charlescal203963.html

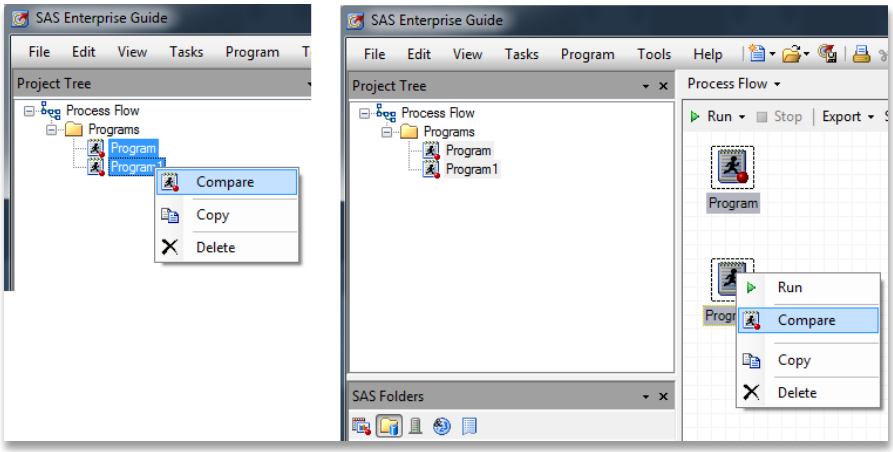

Figure 27: Pop-up Menu Items for Comparing Any Two Selected Programs

PREVIEW UNSAVED CHANGES

What am I about to do? I try to instill in my kids the responsibility to think about the repercussions of their actions before making a decision (though it often feels like it is falling on deaf ears). In the programming world, we (hopefully) review code *before* we push it.

When using the Program History feature, the Changes button enables you to preview *uncommitted* changes. You can easily review any changes you have made to your tracked program since the last committed version. For more information, see the **Preview Changes** section.

However, even if you turn off the Program History feature, the Changes button is still available. In this case, the Changes button enables you to review *unsaved* changes.

Enterprise Guide indicates changes have been made to a program in the Program Editor by appending an asterisk to the Program tab. For an example, see **Figure 28**. To review your unsaved changes, click **Changes** on the contextual toolbar. The current program and the last saved version of that program are then opened into and compared by your third-party file comparison tool. For an example, see **Figure 23**.

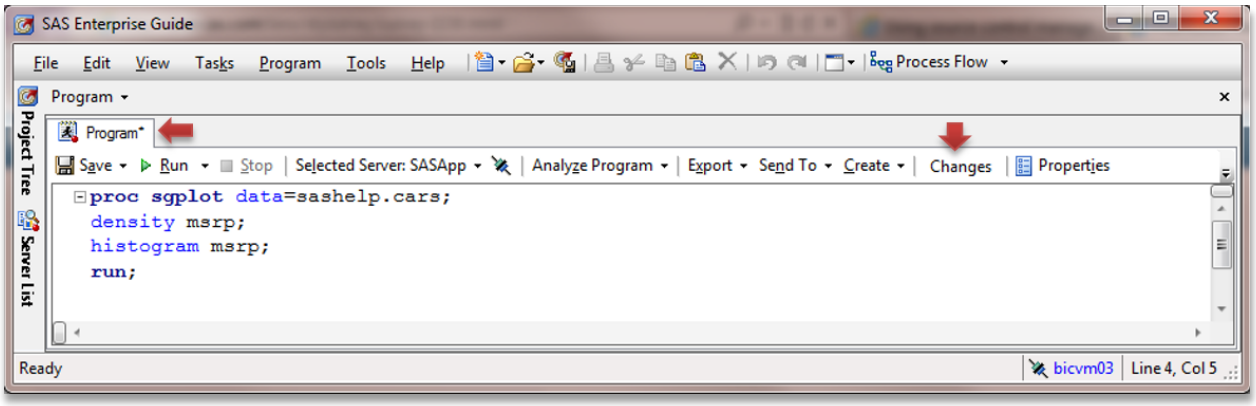

Figure 28: Asterisk Indicating Unsaved Changes and Reviewing Changes Button

CONCLUSION

We hope you have a better understanding of the differences between an embedded program and an externally referenced program. We have shown you how to track changes to an external program using Git in current versions of Enterprise Guide. When you get the chance to try our next release of Enterprise Guide, you will also be able to commit changes to programs, view a history of your changes, and view a history of your commits. We hope these features will help you be more productive and give you better peace of mind. If you have any suggestions or comments, give us a piece of your mind! You can find our contact information below. Go forth and version!

REFERENCES

BrainyQuote. Available at http://www.brainyquote.com/quotes/quotes/c/charlescal203963.html.

RECOMMENDED READING

- Git. Available at http://git-scm.com/.
- *The SAS Dummy*: "Using source control management with SAS Enterprise Guide," http://blogs.sas.com/content/sasdummy/2012/10/29/scm-with-sas-eg/.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors:

Joe Flynn
100 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
joe.flynn@sas.com
http://www.sas.com

Casey Smith
100 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
casey.smith@sas.com
http://www.sas.com

Alex Song
100 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
alex.song@sas.com
http://www.sas.com