

Remembering the Past

Using SAS® Keyboard Macros to Enhance Documentation

Pete Lund
Looking Glass Analytics
Olympia, WA



Who Needs Documentation?

- How many times have you looked at line after line of code and wondered what it did?
 - ...and that's for what you wrote yesterday
- Good in-line documentation is essential for code that is...
 - maintainable
 - understandable
 - “hand-offable”



Adding Consistency to Comments

- If my comments are always formatted the same, I'm more likely to add them and thus have a clue what I did
 - That makes it maintainable
- Those consistent comments also make it easier for someone else to read the code
 - That makes it understandable
- If they (and I) can better understand it, they are more likely to be able to take it away from me someday
 - That makes it hand-offable



3

SAS Keyboard Macros

A Tool for Consistent Comments

- As in some other software packages, SAS offers a means to “capture” and “play back” keystrokes and editor functions
- The result is a keyboard “macro” that replays what you specify
- These “macros” are not related to the SAS Macro Language



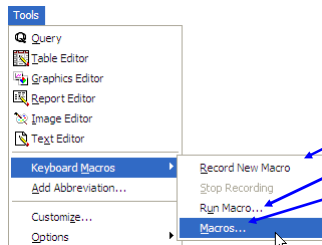
4

Four Documentation-Related Macros

- Today, we'll create four macros to assist in our documentation – they will...
 - add a line for comments
 - add a comment line which is preceded by a dated header – for program changes
 - comment out a selected block of code, adding the current date
 - place a program header template at the top of the program editor

Working with Keyboard Macros

- Click on Tools...Keyboard Macros

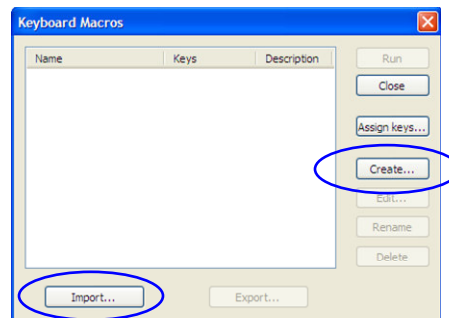


From there you can choose to

- Record a new macro
- Run an existing macro
- Open the Keyboard Macros window to manage existing macros

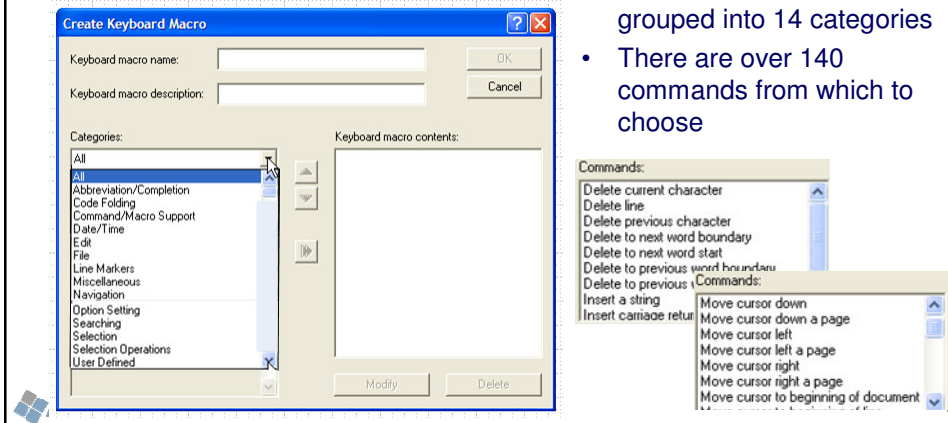
At First, the Keyboard Window is Pretty Dull

- The only things you can do initially are
 - Create a new macro
 - Import a saved macro



Creating a New Macro

- Macros are built by selecting a series of commands to be executed
- Macro commands are grouped into 14 categories
- There are over 140 commands from which to choose



Before We Begin

Build it or Record It?

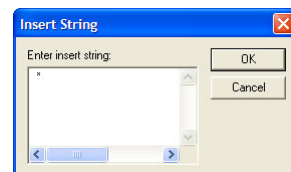
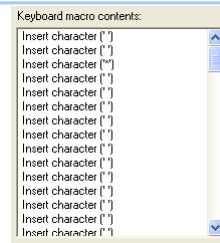
- Keyboard macros can be “recorded” or “built” command by command
- Our first macro will write a comment line to the editor, like this



- Let's go to SAS and see which way works best

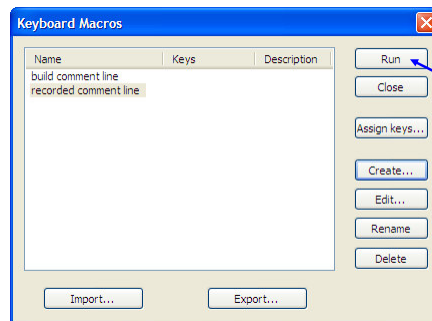
“Built” Usually Wins

- As we saw, the recorded macro had an entry for every keystroke
 - Can make for very difficult editing
- The built macro produced the same results, but with a single command
 - To edit, simply select the command in the macro editor and hit “Modify”



Using Your Macros

- There are two ways to invoke the macros
 1. Hitting “Run” in the Keyboard Macros or Run Keyboard Macro window
 - Not very practical to have to open this window each time you want to run a macro



Hit “Run” and the macro will execute

Major downside: the window closes after “Run” is pressed

Using Your Macros

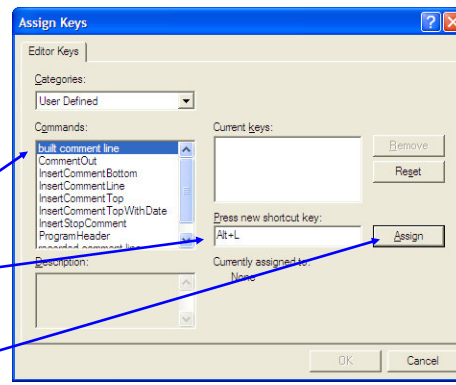
- There are two ways to invoke the macros
 1. Hitting “Run” in the Keyboard Macros window
 2. Assign a keystroke to the macro

Rather than “Run”, hit the “Assign Keys” button



For the “build comment line” macro we’ve assigned “Alt-L”

Click “Assign” to assign the key



Our First Macro

The Comment Line

- The simple example we did has some problems
 - It inserts the line wherever the cursor is, even in the middle of another line
 - We have to move the cursor into the right position to begin entering our comment
 - We have to make sure that overstrike is on or the right side of our comment keeps moving
- Our macro will take steps to remedy all of these
- Each step might require more than one macro command

Step #1 was
the comment
line itself

Our First Macro

Step #2 – Get Out of the Way!

- A new line is inserted wherever the cursor is, even in the middle of another line
- Solution – to get us ready to add the comment:
 - Move to the end of whatever line we're on
 - Insert a carriage return
 - Move the cursor to column 1

Keyboard macro contents:

```
Move cursor to end of line
Insert carriage return
Move cursor to column 1
Insert the string " *
```

This might seem strange, but let's go to SAS and see how all this works together...

Our First Macro

Step #3 – Getting Back into Position

- We have to move the cursor into the right position to begin entering our comment
- Solution – to place the cursor at the right place
 - Move the cursor back to the beginning of the line
 - Move the cursor over four spaces to the right, putting it one space over from the leftmost “*”

Step #2 commands are here, then...

Insert the string " *

Move cursor to column 1

Move cursor right

Move cursor right

Move cursor right

Move cursor right



There is a “Move cursor to the beginning of line” command – why not use that here?

Again, let’s go to SAS and see...

15

Our First Macro

Step #4 – Setting Overstrike Mode

- We have to make sure that overstrike is on or the right side of our comment keeps moving
- Solution – to make sure we don’t mess up the edge
 - Turn the overstrike mode on

Step #1,2,3 commands are here, then...

Set overwrite mode

There is also a “Toggle insert/overstrike mode” command that we could have chosen – why not?



Back to SAS now for the final result...

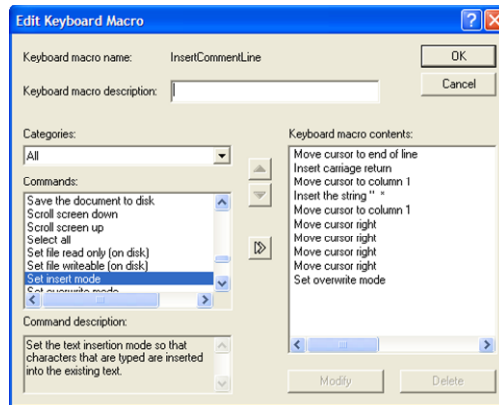
We have no way of knowing what mode is currently set, so we might be turning overstrike on or off.

16

Our First Macro

What Did We Do?

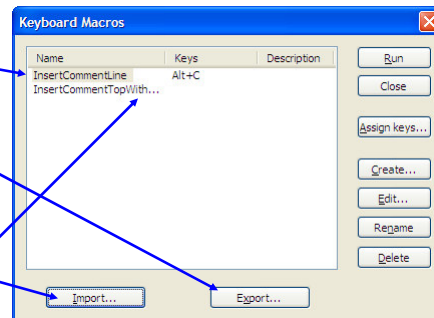
- Our macro ended up with 10 steps



- Move the cursor to the end of the current line to prevent clobbering what's there
- A carriage return to get below the line we're on
- Move to column 1, just in case auto-indent is on
- Insert our comment line
- Move back to column 1
- Move the cursor over to the right four spaces
- Turn overstrike mode on

This is So Cool – Can I Share It?

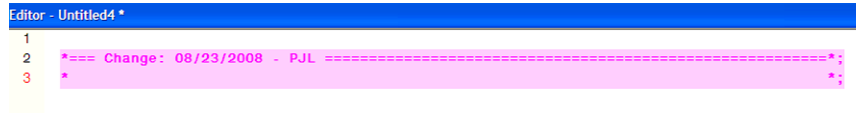
- Keyboard macros can be exported and imported
 - Export
 - Click on the macro name
 - Click on Export...
 - A Save dialog will pop up from which you can save the macro (as a 'kmf' file)
 - Import
 - Click on Import...
 - An Open dialog will pop up and you can choose the macro to open
 - It will appear in the list, with no key assigned



Our Second Macro

Marking Changes to Your Code

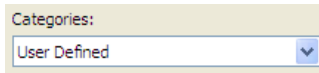
- A very common task, that should be well documented, is making changes to existing code
- We want to know when the code changes were made, so we'll add the current date
- We still want a comment line, so we'll reuse our first macro



```
1
2  *==== Change: 08/23/2008 - PJL =====
3  *
```

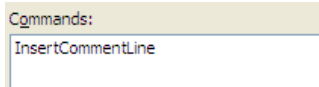
Augmenting Macros the Easy Way

- Note that your user written macros are a category of keyboard macro commands



Categories:
User Defined

- Since we saved the comment line macro, it can be included with a single click into the code change macro



Commands:
InsertCommentLine

- Think about reusing macros
 - Create them from the simple to the complex

Our Second Macro

Adding Dynamic Content

- We'll use very similar techniques to the first example, but...
- ...to get the date, we need to break up the text
 - a static portion: *=== Change :

– the date segments:

- month, followed by a slash
- day, followed by a slash
- year, followed by a slash

```
Insert the string " *=== Change: " at
Insert current month index
Insert character (/)
Insert current day of month
Insert character (/)
Insert current year
Insert the string " - PJL =====
```

– another static portion: - PJL =====* ;

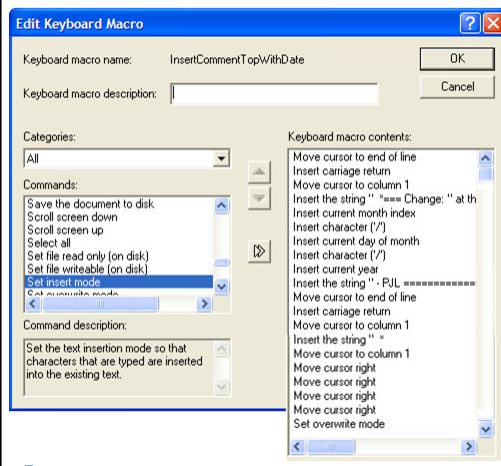


21

Our Second Macro

What Did We Do?

- Our new macro has 20 steps



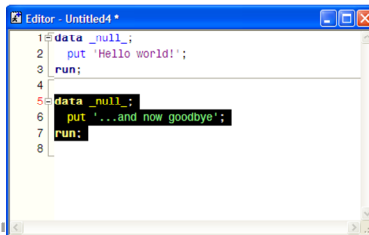
- Move the cursor to the end of the current line to prevent clobbering what's there
- A carriage return to get below the line we're on
- Move to column 1, just in case auto-indent is on
- Insert the first part of the change string
- Insert the current month
- Insert a slash and the current day
- Insert a slash and the current year
- Finish off the change string
- Add the comment macro

22

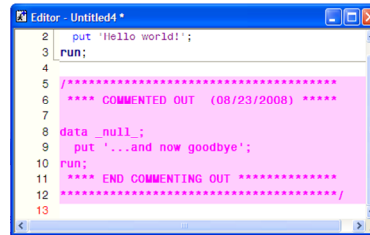
Our Third Macro

Commenting Out a Block of Code

- Another common task, that is not often documented, is commenting out a block of code
- We'll create a macro that will take a selected block of code and...
 - Add block comments around it
 - Insert the current date



```
1 data _null_;
2 put 'Hello world!';
3 run;
4
5 data _null_;
6 put '...and now goodbye';
7 run;
8
```



```
2 put 'Hello world!';
3 run;
4
5 *****
6 **** COMMENTED OUT (08/23/2008) ****
7
8 data _null_;
9 put '...and now goodbye';
10 run;
11 *****
12 **** END COMMENTING OUT *****
13
```

23

Our Third Macro

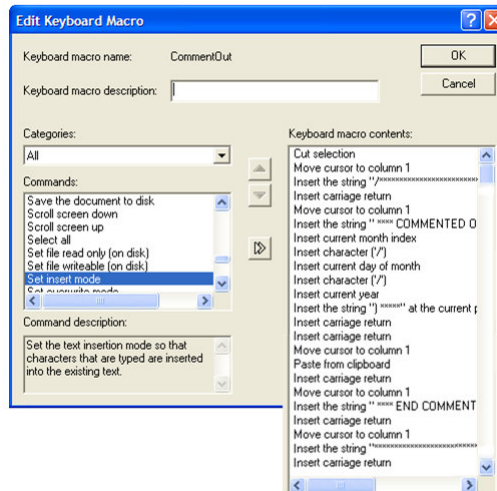
Cutting and Pasting Text

- How do we put the “end of comment” lines at the end of the selected text?
 - We don't know how many lines there are, so we can't “move” the cursor
- Four simple steps to get the comments around the selected text
 - Cut the selected text to the paste buffer
 - Add the top portion of the comment block, with techniques we've been using
 - Paste the selected text back
 - Add the bottom portion of the comment block

24

Our Third Macro

What Did We Do (in just 23 steps)?



Creating a Program Header

- This may look like the most challenging of the macros
- It's actually the simplest
- We've had macros with 10, 20 and 23 commands
- This one has 1



```
1 /* ..... */
2 /* program: ..... */
3 /* programmer: Pete Lund ..... */
4 /* (360) 528-8970 ..... */
5 /* date: ..... */
6 /* project: ..... */
7 /* contact: ..... */
8 /* purpose: ..... */
9 /* ..... */
10 /* ..... */
11 /* ..... */
12 /* ..... */
13 /* incoming: ..... */
14 /* ..... */
15 /* ..... */
16 /* ..... */
17 /* outgoing: ..... */
18 /* ..... */
19 /* ..... */
20 /* ..... */
21 /* macros: ..... */
22 /* ..... */
23 /* ..... */
24 /* ..... */
25 /* format - permanent: ..... */
26 /* ..... */
27 /* ..... */
28 /* ..... */
29 /* macro variables/parameters ..... */
30 /* ..... */
31 /* ..... */
32 /* ..... */
33 /* notes: ..... */
34 /* ..... */
35 /* ..... */
36 /* ..... */
37 /* changes: ..... */
38 /* ..... */
39 /* ..... */
40 /* ..... */
41 /* ..... */
```

Conclusion

- Documenting code is an often tedious, but always beneficial exercise
- Consistently formatted comments make code easier to read
- Keyboard macros give us tools to more easily achieve that consistent appearance
- With 14 categories of commands and almost 140 commands, there's a lot more to do than we've seen today

Author Contact Information

Pete Lund

Looking Glass Analytics
215 Legion Way SW
Olympia, WA 98501

pete.lund@lgan.com
(360) 528-8970
www.lgan.com

