Using Simple Code to Enter Complex Data  
from the DATA Step to SAS/AF®  
By Gui Zupko, US Census Bureau, Washington, DC

ABSTRACT

When data comes from many sources, programmers must decide how to capture that data. However, with the DATA step of PC Base SAS® code, we can make an application that is so user-friendly that even people not experienced with SAS can use it. We can create different options to help our users enter data. First, we will enter data into in the DATA step. Then, we will show how that same program works with macros with only slight alterations. Next, we will demonstrate the effectiveness of %window, using it as a tool for users with no SAS® experience. Finally, we will finish with SAS/AF® and how that application can be employed so that users with no SAS® experience can use it.

INTRODUCTION

With the introduction of new SAS® products, SAS/AF® has been going the way of the dinosaur. However, SAS® users have consistently demanded its inclusion in every version SAS® has released for two reasons. First, some users cannot afford the new products and must rely on the Base SAS® license. Second, SAS/AF® is known and comfortable. SAS® 9.2 is the first version of SAS® in 8 years to start supporting SAS/AF®, and according to Lynn Curley of the SAS Institute, SAS/AF® is here to stay.

Although SAS/AF® has limitations, there are some advantages as well. First, SAS/AF® is available with the Base SAS® license. Second, it is a great way to organize different SAS® programs in one place. Third, the GUI interface is so user-friendly that anyone with limited computer skills is able to use it. Finally, the SCL that SAS/AF® uses is similar to the SAS® programming language, allowing SAS® programmers to easily learn the code.

At a recent team-building exercise, our team decided to do a survey. We needed to analyze the data, which was finding out how many licks it takes to get to the center of a Tootsie Pop. In order to complete imputation and other statistical analysis, we needed demographic information such as gender, the color of the Tootsie Pop, the personality color from the True Colors class taken as part of the team building exercise, and how many months they worked at the Census.

One important note I need to mention up front is that I will be using the sasuser libname for testing purposes. Later on in this paper, it will become possible for other users to enter information. For this to be possible, the users will need access to their own program of SAS® and the program will need to be stored on the network drive.

We will be using the following DATA set:

```sas
data sasuser.database;
input Employee $;
cards;
1
2
3
4
5
run;
data sasuser.database;
  format employee $2. gender $1. color_pop $7. months 4. color $7. licks 4. refusal $3. dates $6. analyst $30.;
```
set sasuser.database;
run;

The following information was received:

<table>
<thead>
<tr>
<th>employee</th>
<th>gender</th>
<th>color</th>
<th>color_pop</th>
<th>months</th>
<th>licks</th>
<th>refusal</th>
<th>dates</th>
<th>analyst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>M</td>
<td>Blue</td>
<td>Red</td>
<td>54</td>
<td>268</td>
<td>No</td>
<td>1/19</td>
<td>Gui Zupko</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>Green</td>
<td>Orange</td>
<td>144</td>
<td>343</td>
<td>No</td>
<td>1/19</td>
<td>Jan Lattimore</td>
</tr>
<tr>
<td>3</td>
<td>M</td>
<td>Green</td>
<td>Yellow</td>
<td>6</td>
<td>10</td>
<td>No</td>
<td>1/25</td>
<td>Brian Hewitt</td>
</tr>
<tr>
<td>4</td>
<td>M</td>
<td>Orange</td>
<td>Brown</td>
<td>5</td>
<td>383</td>
<td>No</td>
<td>1/25</td>
<td>Scott Leighty</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>Orange</td>
<td>Brown</td>
<td>6</td>
<td>875</td>
<td>No</td>
<td>1/26</td>
<td>Kandace Pickeral</td>
</tr>
</tbody>
</table>

ENTERING THE DATA

Since we do not license any of the SAS® products except for Base SAS®, we must program ourselves. That gives us four options to use: the DATA step, macros, %window, and SAS/AF®. Once we have the requirements needed, we can then choose which of these four options best suit our agency’s needs.

- DATA STEP

Creating a database with this method to house this information is not user-friendly at all. All options must be hard-coded in, and most options can only be written after all the data is known. For example, to code in just the first line in the DATA step, the following code would be written:

data sasuser.database;
set sasuser.database;
if employee='1' then do;
gender='M';
color='Blue';
color_pop='Red';
months='54';
licks='268';
refusal='No';
dates="1/19";
analyst='Gui Zupko';
end;
run;

Doing this for multiple employees, even just five, would require at least 50 lines of code and plenty of writing. If I really only had SAS® to use without macros or other more advanced tools, I would write the data in Excel and import it into SAS®. However, a SAS® programmer with basic knowledge of SAS® would be able to enter the data.

- MACROS

There are several macros that can be used here. An actual macro itself would probably be the easiest to both maintain and enter data. The next code would probably look like this:

%macro Enter_data(employee, name, gender, pop, months, color, licks, refusal, dates);
data sasuser.database;
set sasuser.database;
if employee="&employee" then do;
gender="&gender";
color="&color";
color_pop="&pop";
refusal="&refusal";
dates="&dates";
analyst="&analyst";
end;
run;
months=&months;
licks=&licks;
refusal=&refusal;
dates=&dates;
analyst=&name;
end;
run;
%mend Enter_data;
%Enter_data(1, Gui Zupko, M, Red, 54, Blue, 268, No, 1/19);
%Enter_data(2, Jan Lattimore, F, Orange, 144, Green, 343, No, 1/19);
%Enter_data(3, Brian Hewitt, M, Yellow, 6, Green, 10, No, 1/25);
%Enter_data(4, Scott Leighty, M, Brown, 5, Orange, 383, No, 1/25);
%Enter_data(5, Kandace Pickeral, F, Brown, 6, Orange, 875, No, 1/26);

So, we are able to enter in data for all five employees with just 20 lines of code, where before it would have been 50. Of course, if we have more than five observations to add in, we only have to add in 1 line per observation instead of 10.

There are still several problems with this approach. Although I have resolved the amount of code needed to be written, only an intermediate SAS® programmer would be able to enter the data. So although we have made the code more efficient, it is now less user-friendly. Also, the helpful color hints that SAS® gives, such as in the DATA step example above, no longer function within the macro except for Macro commands, denoted by the % sign.

But, what if we had more than just those five users who answered our Tootsie Pop survey? Is a programmer's best use of time entering data into SAS datasets? In not, we now not only have to worry about our programming skills, but the skills of our users. Will the user have enough proficiency in SAS® to design the new line of code into the program?

We can make this new code a bit more user-friendly by using the %let statement. This puts all of the variables at the beginning of the program for the user's convenience instead of at the end. It also clearly defines which variables are being updated by the specific macro. The following lines have been changed:

%let employee=1;
%macro Enter_data;
%Enter_data;

Notice how there are no () marks near the macro name. The corresponding %Enter_data will also not have any () marks. See appendix A for the complete source code for testing. This way, I can save the SAS® program on a shared server and allow the users to run the program themselves. Although this allows users to enter the data themselves, they will need at least a little training on how to enter in the information using this method. Also, any data errors that were entered in would not be caught. Edits would have to be run after the users entered the data to catch possible data entry errors.

- %Window

The problem with the macro approach is that a user unfamiliar with SAS® cannot enter in the data. %Window was my first foray into making SAS® programs more user-friendly. If a regular user could enter in the data, then the program would be much time efficient for me, the programmer. One way is to use %let statements at the beginning and have the user change each individual line. However, the problem is that users might forget or skip a line and the resulting database would have the wrong data. %Window reduces the likelihood of such errors:

%window gui
  #5 @10 "Enter in SASUSER DATABASE Data " color=blue
  /*The number after the # denotes how far down the line appears, the number after the @ denotes how far over the line appears*/
  #7 @5 "Enter Employee Number " employee 2 attr=underline autoskip=yes
  /*The word after the quoted words is the name of the macro.*/
/*The number is how long that macro can be*/
/*The third word is the attribute those macros will have in the %window.*/
/*The fourth word turns on the autoskip feature*/
#8 @5 "Enter Name " Name 30 attr=underline autoskip=yes
#9 @5 "Enter Your Gender " gender 1 attr=underline autoskip=yes
#10 @5 "Enter Your Tootsie Pop Color " pop 8 attr=underline autoskip=yes

#11 @5 "Enter How Long You Have Been "
#12 @7 "At The US Census Bureau " months 3 attr=underline autoskip=yes
#13 @5 "Enter Your Personality Color " color 8 attr=underline autoskip=yes

#14 @5 "Type X If You Refuse To Suck A Tootsie Pop " refuse 1
attr=underline autoskip=yes
#15 @5 "Enter In Number of Licks to Center of Tootsie Pop "
color=green licks 3 attr=underline autoskip=yes color=green
#16 @10 "Hit Enter To Run" color=red;

%macro Enter_data;
%let employee=%str();
/*The %str means that when the program is run multiple times, the macro is erased. When you want to keep a macro through multiple runs of the program, do not include the %str;/*
%let name=%str();
%let gender=%str();
%let pop=%str();
%let months=%str();
%let color=%str();
%let refusal=%str();
%let licks=%str();
%display gui;
/*The %display is where in the program the %window is shown.*/
%let dates=%substr(&sysdate,1,5);
%let refuse=%upcase(&refuse);
%if &refuse=X %then %let refusal=Yes;
%else %let refusal=No;
data sasuser.database;
 set sasuser.database;
  if employee="&employee" then do;
    analyst="&name";
    gender="&gender";
    color_pop="&pop";
    months="&months";
    color="&color";
    licks="&licks";
    refusal="&refusal";
    dates="&dates";
  end;
run;
%mend Enter_data;
%Enter_data;

This code, when run, creates an interactive window. After entering in the data for employee 1, the view would look like Fig. 1:
Once all of this data is entered, the sasuser.database will then look like this:

<table>
<thead>
<tr>
<th>employee</th>
<th>gender</th>
<th>color</th>
<th>color_pop</th>
<th>months</th>
<th>licks</th>
<th>refusal</th>
<th>dates</th>
<th>analyst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>m</td>
<td>blue</td>
<td>red</td>
<td>54</td>
<td>268</td>
<td>no</td>
<td>1/19</td>
<td>gui zupko</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>Green</td>
<td>Orange</td>
<td>144</td>
<td>343</td>
<td>No</td>
<td>1/19</td>
<td>Jan Lattimore</td>
</tr>
<tr>
<td>3</td>
<td>M</td>
<td>Green</td>
<td>Yellow</td>
<td>6</td>
<td>10</td>
<td>No</td>
<td>1/25</td>
<td>Brian Hewitt</td>
</tr>
<tr>
<td>4</td>
<td>M</td>
<td>Orange</td>
<td>Brown</td>
<td>5</td>
<td>383</td>
<td>No</td>
<td>1/25</td>
<td>Scott Leighty</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>Orange</td>
<td>Brown</td>
<td>6</td>
<td>875</td>
<td>No</td>
<td>1/26</td>
<td>Kandace Pickeral</td>
</tr>
</tbody>
</table>

This makes it so that any user can run the program and then create their own observations. However, it is important to note some important problems. First, notice how employee 1 did not capitalize the data values. In some cases, such as with the variables months and licks, this does not matter. However, I intend to use the data for statistical purposes. If I group according to color, blue would be considered different from Blue. So, these types of programs need to have different types of edits built in to ensure the correct capture of data. One type of edit that I built into the program is by using the refuse macro. I capitalized any data entered into that macro, then forced that macro to be capitalized. So, even if an user entered in the letter x, the macro would be capitalized and X's would be recorded to catch the user's true meaning.

It is also interesting to note that this took 42 lines to write, and adding more edits to the data increases the amount of lines needed. Logic is also needed to enter in these edits in many cases, using If→Then statements and even loops. So, for few observations, it does not make sense to use the %window option. However, if less savvy SAS users need to enter in the information themselves instead of passing it on to the more experienced SAS users, the increase in user-friendliness would justify the increased programming.

Another problem that I encountered with %window is that users will clear the program editor and then save the program. That means that the user will save a blank program editor instead of the program that has been laboriously created. The only way to stop this from happening is if the program is made read-only. That means future changes to the program are more difficult.

In order to create %window, an intermediate SAS® user with a firm grasp of both macros and %window is needed. Interestingly enough, the %window program defining the database is quite similar to the macro program defining the database. However, if the %window code is not written correctly, debugging the code is extremely difficult since the log is usually not helpful in determining where the problem exists. I write the %macro code first, and then determine whether to use %window.
The problem with %window as an interface is twofold: first, although you can change the color and placement of the words, nothing else can really be changed; second, it takes time to write up the location and placement of the words and assign macros to them. I was happy with %window for a couple of years until I discovered SAS/AF®. However, the SCL code that is required for SAS/AF® took me a while to learn, and I only felt comfortable with it after taking a class on it. My problem with SAS/AF® is that I could write SAS® and SCL code comfortably, and I could easily make the GUI displays, but getting those two mediums to communicate was difficult. Debugging SAS/AF® is much like debugging %window, going over each line of code for a mistake.

To create a frame, the first thing that is needed is a catalog. Since we are putting our data in sasuser, then that is a good place for our catalog. To create the catalog, we enter the SAS window Explorer and double click on the library of our choice (sasuser in the example), then right click within the library and select New. We need to name our catalog, and for this example, we are using the name Tootsie_pop. Double click on the catalog Tootsie_pop to open the catalog. Right click again and select New, and a new dialog box will open. Choose the selection named ‘Frame’, and then we can start to create our GUI display.

Now that we have reached a blank frame, we have to consider how to create it. We know that there are the following variables to consider:

- employee
- gender
- color
- color_pop
- months
- licks
- refusal
- dates

so making the frame should not be too difficult. While creating this frame, however, we have to choose which components to use. Choosing the correct component will make creating edits and increasing user-friendliness easier. Since employee, gender, and color are character variables with a set amount of options, a combobox would be my first choice. That gives the user the choice of typing in their option or using a point and click method. However, they can still type in the wrong option, so you might still need to use an edit. Sucker color, which only had five options, is ideal for using a radiobox. The combobox would still be appropriate here, but for demonstration purposes, I chose to use the radiobox. Months and licks are both numeric variables, so they are ideal candidates for using a spinbox. This allows us to create a max and min on the variables and choose what integers are used. Finally, we have dates, which can be created using many different components. Just for fun, I used a text box here to show potential problems that can appear, although I would probably use radioboxes here as well. Finally, refusal can only be a yes or no option, which makes it an ideal candidate for the checkbox. After creating the needed comboboxes, radiobox, spinboxes, and textentry, and then modifying the border titles, adding items, and setting mins and maxes for months and licks in the properties box, the product is now workable. Although these can be arranged in whatever order is desired, my finished product looks like Fig. 2.

![Figure 2](image_url)

The options above allow individual users to input their response information themselves, instead of pester me with the information. The GUI display above is such that anyone with basic computer skills is able to enter in the information. While SAS/AF® is much more user-friendly than all the other options above, it is still clunky and inefficient compared to today’s technology.

I mentioned before that the textentry here can be problematic. The problem with textentry for dates is that any information is saved, regardless of how correct it is. For example, if a user wrote ‘today’ in the textentry box, then the program would allow that. However, even if the user wrote down a date, what format would that user use? The user could write virtually anything, and programming edits for this would be challenging. That is why I suggest using radioboxes, especially for dates, or if I wanted to save the current date, use the today() function behind the scenes and not even ask the user the date.
Now that the user has entered in the data, they need to be able to tell the program that they are finished entering data. We have two options to consider: having the data save automatically into the dataset as soon as it is entered or only save once all data has been added. The amount of coding needed for the first option is significantly larger than the second and requires greater logic. The second option is easy to code and requires little logic. It simply requires the user to click on a pushbutton that would save the data. Another pushbutton will allow the user to exit the application. The application would look like Fig. 3:

![Fig. 3](image)

Now, there only remains the coding in SCL. The ‘Save Employee’ pushbutton is not enabled, which means that the user cannot use it. We can enable the pushbutton by using the following code:

```
combobox1:
if combobox1.selecteditem='' and combobox2.selecteditem='' then pushbutton1.enabled='Yes';
return;

combobox2:
if combobox1.selecteditem='' and combobox2.selecteditem='' then pushbutton1.enabled='Yes';
return;
```

This code means that as soon as information appears in the ‘Employee’ and ‘Gender’ comboboxes, the ‘Save Employee’ pushbutton will no longer be greyed out and users will have the ability to save the data. In a real program, we would include all of the variables, except for the number of licks and the months at the U.S. Census Bureau, before we would enable the ‘Save Employee’ pushbutton. Since these two fields already have a 1 in them, we would want to count new employees and those who bit the Tootsie Pop in half in the first bite.

In order to allow the users to save into an actual SAS DATA set, we need to code the ‘Save Employee’ pushbutton, which is pushbutton1. The code will have three sections: 1) creating the macros, 2) saving the data in a SAS dataset, and 3) blanking the data gathered.

```
pushbutton1:
call symput('employee',combobox1.selecteditem);
call symput('gender',combobox2.selecteditem);
call symput('months',spinbox1.text);
call symput('licks',spinbox2.text);
call symput('refusal',checkbox1.selected);
call symput('color',combobox3.selecteditem);
call symput('color_pop',radiobox1.selecteditem);
call symput('dates',textentry1.text);
submit continue;
data sasuser.dataset;
set sasuser.dataset;
if employee="&employee" then do;
```

7
We need to create macros in order to make SAS® code out of SCL code. The call symput function is the same as in SAS® code, creating a macro that can cross the boundaries between SAS® and SCL code. Once these macros are created, we can use them in SAS® code.

By using the submit continue command in SCL, we are able to use SAS® code. Notice how this SAS code is exactly the same as the code in the macros and %window section. The problem with embedding SAS® code in this SCL code is that only those familiar with SCL code are able to modify or maintain the code. I found that by using a %include function in the submit continue function allows SAS® users the ability to add or change the program. So, if a variable name changed in the dataset, a regular SAS® user, even one unfamiliar with SCL code, would be able to maintain the code. For me, the variable changing was the year.

Finally, we do not want data from one employee to end up with another employee. So the last third of the code blanks out all the data and sets it up for the next employee to enter data.

Now that we have our frame, how do the users access it? We have two options at our disposal. First, we could set it up and allow users to open the catalog and then right click on the frame, then selecting ‘Run’. The problem with this approach is that most users would need training in order to use this method. I prefer to make a copy of my configuration file, rename it ‘Tootsie.CFG’, and transfer it to the directory that will be used. I usually create a data folder in that directory to house all the background work. Next, I copy the SAS shortcut icon and rename it ‘Tootsie Pop Database’, and change it to open my new config file on the network drive and the Tootsie Pop frame that I have created. The resulting product looks like Fig. 4:
At the bottom of the Tootsie.CFG, I added the following code:

```
set guizer 'C:\NESUG 2010\Data'
-initcmd "af c=guizer.tootsie_pop.entry.frame; wstatusln off; wwindowbar off; command close;
toolclose; awsmaximize on"
-awsdef 0 0 100 100
-awstitle 'Suckers'
-splashloc "C:\NESUG 2010\Data\Census.bmp"
-noawsmenu
-sascontrol nominmax nosystemmenu
-initstmt 'footnote "as of &sysdate"';
```
Only make changes to the config file copies made. We do not want to accidentally make changes to the actual config file and not be able to use SAS at all. Now, when I double-click on the Tootsie Pop Database icon, the following image appears:

![U.S. Census Bureau](Fig. 6)

while SAS® is loading. Finally, we are rewarded for our hard programming work with the program appearing in Fig. 7:

![Tootsie Pop Data Entry](Fig. 7)

Because of the additions we added in the Config file, SAS® automatically opens to a full screen. Although we could turn off the menu options in the upper left corner, I prefer to leave them on in case the program halts unexpectedly. If we turn it off, the user has no way to exit SAS® except by terminating it from the outside of the program. We also have the title ‘Suckers’ in the upper left hand corner.

So, even though SAS/AF® is the most compelling of our options, it has some problems as well. First, since SAS started developing other products which have more point and click options, they have stopped supporting SAS/AF® altogether. This attitude has changed recently based on user preferences beginning with SAS® 9.2. When I went to the SAS® Global Forum in 2009 about how to get SAS/AF® to work more efficiently, no one in the SAS® Support Area could help me. Second, it takes time to code and set-up the GUI interfaces, and for small projects that are used every once in a while, it does not make
sense to spend the time developing this code. Third, code and logic problems in SAS/AF® are difficult and time-consuming to find. I wrote the code in Appendix B in about 30 minutes, but then had to spend about 3 hours debugging it and checking it for errors. For minor projects, that just does not make sense. However, if we are building a SAS® database that incorporates the data gathered with statistical analysis, SAS/AF® might be the way to go. Finally, SAS/AF® is only effective when the users are all connected via a network. Writing this code onto each machine individually would be horrific.

CONCLUSION

We have gone over several ways of collecting data from individuals. Now, as programmers, we have to decide which option is best for our users’ needs. For coding in data, I would never recommend the first option of the DATA step unless the programmer did not know macros at all. Personally, I code the second option, the macros, first. This is an easy way for me to enter in the data quickly and check for potential problems. If, however, there is so much data that there is too much for any one person, I would then use the macro program that I wrote to create a %window option. For small amounts of data, %window is great. However, if there are many variables, %window does not provide the flexibility to clearly show the user what needs to be done. Then, I would create the %window for short-term use while I created the SAS/AF frame. However, if the user wants greater output options, such as graphs or other forms of statistical analysis, SAS/AF® has the best options for creating a SAS® database. However, the more user-friendly that we make a program, the more coding there is for the programmer.

REFERENCES


ACKNOWLEDGEMENTS

This report is released to inform interested parties of (ongoing) research and to encourage discussion (of work in progress). Any views expressed on (statistical, methodological, technical, or operational) issues are those of the author(s) and not necessarily those of the U.S. Census Bureau.

I would like to thank the U.S. Census Bureau and SAS for this opportunity to write and present this paper. I would also like to thank Jan Lattimore and Brooklyn Lupari, both at the U.S. Census Bureau, who took time to edit and check this paper to make sure that it was factually and grammatically sound. Finally, my thanks go out to the New Employee Group in the Manufacturing and Construction Division of the U.S. Census Bureau who gave me the opportunity for writing the above code and for allowing their names to be used for demonstrative purposes.

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. © indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.

CONTACT INFORMATION

William ‘Gui’ Zupko II
US Census Bureau
4700 Silver Hill Road
Suitland, MD 20746
William.e.zupko.ii@census.gov

APPENDIX A: MACRO CODE WITH %LET FOR TESTING

%let employee=1;
%let analyst=Gui Zupko;
%let gender=M;
%let color=Blue;
%let months=54;
%let color_pop=Red;
%let licks=268;
%let refusal=No;
%let dates=1/19;

%macro Enter_data;
  data sasuser.database;
    set sasuser.database;
    if employee="&employee" then do;
      analyst="&analyst"
      gender="&gender"
      color_pop="&color_pop"
      months="&months"
      color="&color"
      licks="&licks"
      refusal="&refusal"
      dates="&dates"
    end;
  run;
%mend Enter_data;

%Enter_data;

APPENDIX B: SCL CODE WITH PUSHBUTTON TO SAVE FOR TESTING

combobox1:
  link enabler;
  return;

combobox2:
  link enabler;
  return;

combobox3:
  link enabler;
  return;

radiobox1:
  link enabler;
  return;

textentry1:
  link enabler;
  return;

checkbox1:
  link enabler;
  return;

enabler:
  if combobox1.selecteditem='' and combobox2.selecteditem='' and combobox3.selecteditem='' and radiobox1.selecteditem='' and textentry1.text='' and checkbox1.selected='No' then
    pushbutton1.enabled='Yes';
  return;

pushbutton1:
  call symput('employee',combobox1.selecteditem);
  call symput('gender',combobox2.selecteditem);
  call symput('months',spinbox1.text);
call symput('licks',spinbox2.text);
call symput('refusal',checkbox1.selected);
call symput('color',combobox3.selecteditem);
call symput('color_pop',radiobox1.selecteditem);
call symput('dates',textentry1.text);
submit continue;
data sasuser.dataset;
set sasuser.dataset;
if employee="&employee" then do;
    gender="&Gender";
    months=&months;
    licks=&licks;
    refusal="&refusal";
    color_pop="&color_pop";
    dates="&dates";
end;
run;
endsubmit;
combobox1.selecteditem='';
combobox2.selecteditem='';
combobox3.selecteditem='';
spinbox1.text=1;
spinbox2.text=1;
checkbox1.selected='No';
pushbutton1.enabled='No';
radiobox1.selecteditem='';
textentry1.text='';
return;

APPENDIX C: SCL CODE WITHOUT PUSHBUTTON TO SAVE FOR TESTING

Without pushbuttons, we need a new variable, designated locked, to signify which observations a user is currently updating. Also, since the min properties of months and licks set to 1, we need to change the dataset to be such:

data  sasuser.dataset;
    format employee $2. gender $1. color_pop $7. months 4. color $7. licks 4. refusal $3. dates $6. locked $1.;
set sasuser.dataset;
    locked='N';
    licks=1;
    months=1;
run;

The SCL code, naming the objects in the properties box exactly the same as in the data set:

declare list sucker_items={};
length sucker_first sucker_last test xx last_employee 4 varia $12 varia_value $16 varia_number 4;
init:
guiz1='sasuser.dataset';
submit continue;
%macro zippy;
data _null_; set &guiz1 end=last;
    where locked='N';
    if _n_=1 then call symput('sucker_first',employee);
    if last then call symput('sucker_last',employee);
run;
%put 'This is here ' &sucker_first &sucker_last;
%mend zippy;
%mend;
endsubmit;
sucker_first=symgetn('sucker_first');
sucker_last=symgetn('sucker_last');
do i=sucker_first to sucker_last by 1;
  rc=insertc(sucker_items,i,-1);
end;
sucker.items=sucker_items;
last_employee=.;
return;

employee:
  employees=employee.selecteditem;
  put 'This is the last employee' last_employee;
  if last_employee^=. then do;
    employee.selecteditem=.;
    months.text=.;
    licks.text=.;
    xx=last_employee;
    dsid=open(guiz1,'U');
    call set(dsid);
    f=fetchobs(dsid,xx);
    vnum=varnum(dsid,'locked');
    cval=getvarc(dsid,vnum);
    call putvarc(dsid,vnum,'N');
    rc=update(dsid);
    rc=close(dsid);
  end;

  errors=employees;
  test=searchC(sucker_items,employee.selecteditem);
  if test^=0 then do;
    put 'This is the test' test;
    xx=employees;
    dsid=open(guiz1,'U');
    call set(dsid);
    ccf=fetchobs(dsid,xx);
    vnum=varnum(dsid,'locked');
    cval=getvarc(dsid,vnum);
    if cval='Y' then do;
      do while (cval='Y');
        xx+1;
        f=fetchobs(dsid,xx);
        vnum=varnum(dsid,'locked');
        cval=getvarc(dsid,vnum);
        end;
      end;
      call putvarc(dsid,vnum,'Y');
      rc=update(dsid);
      rc=close(dsid);
    end;
  end;
  last_employee=employees;
  return;

updater:
  dsid=open(guiz1,'U');
  f=fetchobs(dsid,xx);
  vnum=varnum(dsid,varia);
if varia in ('Gender', 'Color', 'Color_pop', 'Refusal', 'Dates') then call putvarc(dsid,vnum,varia_value);
else call putvarn(dsid,vnum,varia_number);
rc=update(dsid);
rc=close(dsid);
return;

Gender:
varia='Gender';
varia_value=gender.selecteditem;
link updater;
return;

Color:
varia='Color';
varia_value=color.selecteditem;
link updater;
return;

Color_pop:
varia='Color_pop';
varia_value=color.selecteditem;
link updater;
return;

dates:
varia='Dates';
varia_value=dates.text;
link updater;
return;

refusal:
varia='Refusal';
varia_value=refusal.selected;
link updater;
return;

months:
varia='Months';
varia_number=months.text;
link updater;
return;

licks:
varia='Licks';
varia_number=licks.text;
link updater;
return;

pushbutton1:
if last_employee^=. then do;
dsid=open(guiz1,'U');
call set(dsid);
f=fetchobs(dsid,xx);
vnum=varnum(dsid,'locked');
cval=getvarc(dsid,vnum);
call putvarc(dsid,vnum,'N');
rc=update(dsid);
rc=close(dsid);
end;
_status_='H';
return;