Paper 1042

One SAS To Rule Them All…

William “Gui” Zupko II, Federal Law Enforcement Training Centers

ABSTRACT

In order to display data visually, our audience preferred Excel’s compared to SAS’s charts and graphs. However, to make the necessary 30 graphs in Excel took 2-3 hours of manual work, even having chart templates already created, and also led to mistakes due to human error. SAS graphs took much less time to create, but lacked key functionality that the audience preferred available in Excel graphs. Thanks to SAS, the answer came in X4ML programming. SAS can actually submit coding to Excel in order to create customized data reporting, create graphs or update templates’ data series, and even populate word documents for finalized reports. This paper explores how SAS is used to create presentation-ready graphs in a proven process that takes less than one minute, compared to the earlier process that took hours. The following code will be utilized and/or discussed: %macro(macro_var), filename, rc commands, ODS, X4ML, and VBA (Microsoft Visual Basic for Applications).

INTRODUCTION

SAS programmers often need to decide how best to provide information, especially big data, to their audience. Often, our wants and needs are not even known until something changes. While providing graphs to our audience, I first provided the graphs in Excel and manually updated them with information from SAS. Since the information was dynamic, I was unable to create a template in Excel updating graph series with hard coded cell information. For example, sometimes a graph was for an entire year, which would have hard coded those days. However, sometimes the graph was for as little as a week, meaning that I had to constantly update those Excel graphs. Using the power of SAS, we embarked on a journey to find the best way to present our audience with their critical information.

Since SAS has graphing capabilities, my first attempt was to create SAS graphs directly in Excel. This was extremely easy for me and allowed me to create the graphs and export them via ODS. However, a couple of problems immediately occurred that threw kinks into this plan.

First, when exporting graphs from SAS to Excel, the graphs are not actually created in the Excel file. They are created as .gif or other picture formats that can be set by the SAS programmer, and a link was created in the Excel file looking at these graphs. However, when the Excel file was sent, the images were no longer found with the linking structure and, as a result, the file only showed blank pictures saying “File Not Found”. In order to get around this problem, I would open the picture, copy the image, and then paste it into the correct location. Took some time, but provided a finished product.

Second, since the graphs were .gif, they could be resized, but there was none of the advanced functionality included in those graphs. It was only after the ability to hover over the graph and see that data point’s information that our audience realized this was important, if not critical, functionality. Unfortunately for me, that meant again hand coding tables into Excel, again risking human error and taking 1-2 hours to enter in the information. I was determined to try again.

FORMATTING THE DATA

In order to get the information into Excel, it would need to be formatted. My first effort was to use proc export, which got the data into Excel just fine, but I soon found that data did not come with format that I would like to use. For example, I might want to include specific macros in SAS, but there is no way for a SAS macro to talk to Excel. Since these macros were often user-based, there was no way to have Excel understand what information was being included in its spreadsheets.

Since I needed specific information simply to be available in Excel, not necessarily understandable to anyone but me, I began to tag the data. For instance, I am going to have three total charts in my Excel spreadsheet, so I will need three different tags. The first tag is for “All”, meaning instructors A, B, C, D, E, and F. The second tag is for “ABC”, and the final tag is “DEF”. With the data now tagged, I put all of the data into one data set using the multiple set function. With one data set, I used ODS TAGSETS.EXCELXP to output the data into Excel. In version 9.4, the ODS Excel option also allows the same functionality. These gave me several options that were not previously available:

- Use ODS styles
- Freeze header frame for browsing
- Name worksheets, not just filename
- Put on filters
- And so much more…

When the data was sent to Excel, the following view appeared:

Figure 1. Raw Excel Output

	A	B	C	D	E	F
1		*SESUG Presentation*				
2		*Program Run on 16JUN15*				
3						
4	grapher	time_var	class_da	time_cla	tot_instructo	num_instructo
5	All	04/27/15 - 7:30:00	04/27/2015	7:30:00	6	6
6	All	04/27/15 - 7:45:00	04/27/2015	7:45:00	6	6
7	All	04/27/15 - 8:00:00	04/27/2015	8:00:00	6	6

In column A, I have my tagging variable. I am now able to see variable names, filter the data to my heart's content, and even get a nice title.

However, my audience wants a graph to visualize the data, not a table with thousands of observations. Unfortunately, when I ran this, tagging the data was insufficient to pass all required information. I had two other ways to enter the data that I needed.

First, I can use the title itself to pass along data to Excel. While the title here is somewhat innocuous, what if I want to include in the title the date range that was selected by the user. Also, the variable name tot_instructor might make sense to me, but that variable might be difficult for my audience to understand. I can use the label function in the DATA step to make titles that would be more appropriate for my audience. By using these two options, I then had the following output in Excel:

Figure 2. Formatted Excel Output

	A	B	C	D	E	F
1		*SESUG Presentation*				
2		*04/27/2015-05/01/2015*				
3						
4	grapher	Day/Time	class_da	time_cla	Instructors Assign	Total Instructo
5	All	04/27/15 - 7:30:00	04/27/2015	7:30:00	6	6
6	All	04/27/15 - 7:45:00	04/27/2015	7:45:00	6	6
7	All	04/27/15 - 8:00:00	04/27/2015	8:00:00	6	6

In row two, I now have my date range all set up. Additionally, in row 4, the variable names that will be included in the graphs have also been updated so that I can see that column E is how many instructors are assigned and column F is how many instructors total I can assign.

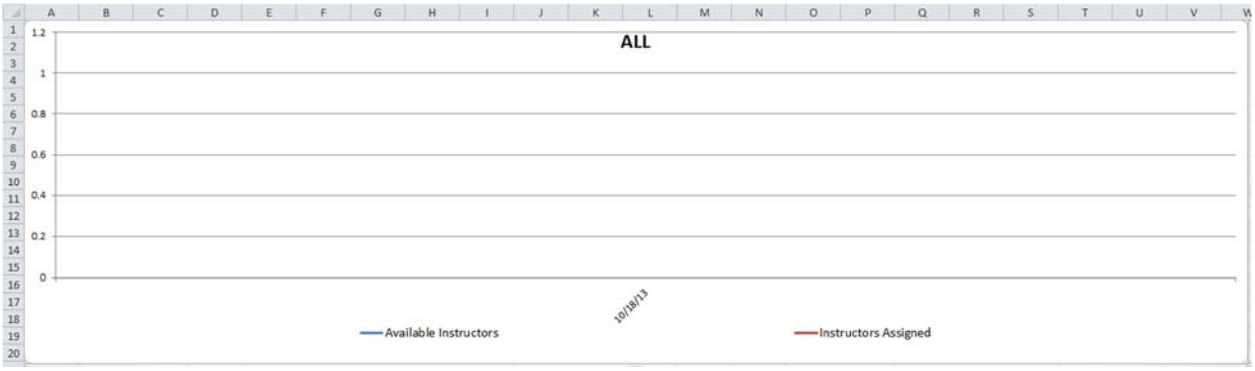
Now, we have our data in Excel, formatted for the look and to provide additional critical information, and are now ready to create our graph!

CREATING THE GRAPH

Since our audience wants Excel graphs, we have two options for creating a graph. The first option is to use Microsoft Visual Basic for Applications (VBA) to create a graph from the ground up. The second option is to create a new file in Excel and create a template graph there, then use VBA to copy the graphs into the Excel file we just created. Either option has its pros and cons, but it really depends on personal preference. If you prefer to code a graph and build it that way, you have that option with VBA. I prefer to build the graph in a template file myself, seeing what it looks like, and then copy that, simply updating the data. I have also found that this decreases maintenance and trouble-shooting costs.

The code for the graph in this example uses two data series and updates the horizontal axis, using a graph from a template. While it does not look like much, it provides the colors for the lines, automatically providing that information for our Excel graph. Below is a copy of the graph template:

Figure 3. Excel Graph Template



With the graph template created, we are ready to go into VBA.

CODING IN VBA

VBA is pretty easy to use, especially with the record macro function. You can switch on record macro, do the action that you want, and then turn off the recording. While this paper will focus on these graphs and procedure, it is very easy to adopt other functionality with the record procedure, even beginners in VBA code.

With the record function, it is easy to make your own functionality, but problems occur knowing where data is located. Using our tags, we are able to start this process. In order to find the first iteration of "All", I used the following code from Hawley, Dave:

VBA Code 1. Finding the Row Number of the First "All"

```
Dim lRow As Long

    On Error Resume Next
    lRow = Application.WorksheetFunction.Match("All", Range("A:A"), 0)
    On Error GoTo 0

    If lRow > 0 Then
    End If
```

This code creates a variable in VBA called lRow, which I can then use later in the code. When creating Excel variables, I always double check by putting information into a blank cell to confirm that it is the number that I need. Once I have the first row, now I need to find the last row. The following code finds that variable:

VBA Code 2. Finding the Row Number of the Last "All"

```
Dim myC As Range
Dim finalrow As Long

 Set myC = Range("A:A").Find("All", , , , , xlPrevious)

  finalrow = myC.Row
```

We now have two Excel variables created, lrow for the first row, and finalrow for the last row. Just remember, for this process to work, like tags have to be next to each other. If All is the first and last observation from a sort, everything would be included.

When I recorded changing my values, I was given the following VBA code:

VBA Code 3. Recorded Code

```
ActiveChart.SeriesCollection(1).Values = "=raw!$E5:$E$156"
```

VBA records things in quotes are direct values and things outside of quotes as variables. The & acts as a concatenation. By making this minor change, instead of pointing to the number 5 for the first row and 156 for the last row, I will be using the Excel variable which will resolve the same way.

VBA Code 4. Modified Code

```
ActiveChart.SeriesCollection(1).Values = "=raw!$E$" & lRow & ":$E$" & finalrow
```

By using the record function, anyone can get the VBA code needed for their creation. Then, once the code is there, it can be modified to search for the specific tags that we included in the SAS code to find where it should be.

VBA MACROS

VBA Code is contained in VBA macros. Just like SAS macros, these essentially store the code until called. While it is possible to write VBA code directly in SAS, I find that it is easier to instead write the code in VBA, then use SAS to call the specific code forward. The pro to writing the code in SAS is that it is easier to make names and other conventions using macros in the SAS code. The pro to writing in VBA is the information is already in Excel and does not have to be figured out where it will be. Depending on the user's needs, either option might be right.

At this point in time, I have created four VBA macros:

1. Move the worksheet containing the Excel graph template to a new file
2. Input data into graph All
3. Input data into graph ABC
4. Input data into graph DEF

Depending on whether you open the Excel raw data first or not will make this first step superfluous. If Excel is not open, then Excel needs to be open before VBA code can run. We can use the x options to open Excel for us. However, we need to turn on the options noxwait and noxsync, because otherwise, SAS will not run anything until Excel is closed again. Then, we can use the x options to open Excel. In order to do this, we need to find the location of the file excel.exe. Just do a search in your program files, copy the location, paste it into SAS in quotes with the x option, and let SAS take control of Excel.

While it would be wonderful for Excel to open as soon as it was called, this often takes some amount of time. In order to make sure that Excel is fully open, I use the following code to make just a slight, five second delay so that I am not trying to run Excel macros before the system is ready.

SAS Code 1. Delay SAS from running immediately

```
data _null_;
rc = sleep(5);
run;
```

SAS has the option of using X4ML programming through DDE, which allows third party software to talk to Microsoft Office products. Since DDE is not a SAS product, if any issues arise using this procedure at this point, then the point of contact would be Microsoft. Also, SAS support did inform me that DDE has been known to cause issues for programmers in the server environment, while it seems stable for the PC environment.

In order to call up DDE, use a filename using dde as a file option, specifically opening "excel|system". Then using the filename that we have created, in this case, sesug23, call forth the code necessary to run. If the raw data file is not already open, then we will need to open that as well. The way my code runs, it is already open, so that section is commented out:

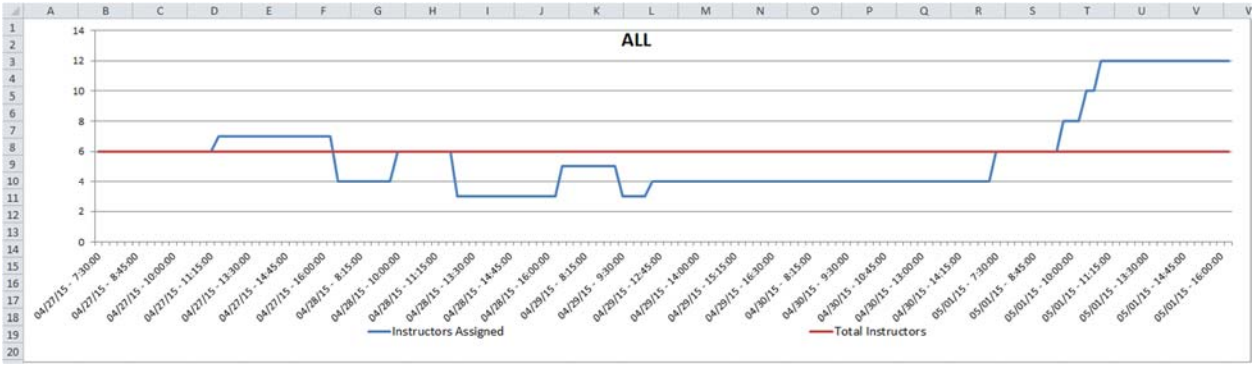
SAS Code 2. Creating the Graphs in Excel

```
data _null_;
    file sesug23;
/*          put "[open(""C:\Users\wezupkoii\Documents\SAS Presentations\First Save
Graphs.xls"")]";*/
            put "[open(""C:\Users\wezupkoii\Documents\SAS Presentations\SESUG Graph
Template.xlsm"", 0 , true)]";
            put "[run(""Move"")]";
            put "[run(""GraphAll"")]";
            put "[run(""GraphABC"")]";
            put "[run(""GraphDEF"")]";
```

So, this code opens the raw data file (commented out), then opens the template file where my graphs and macros are stored. The additional code at the end of that statement makes it open a read-only file, which will protect my template file if things go wrong. The run statement means to call up the VBA macros that are stored in my source file, which then do all the work. However, when moving multiple spreadsheets, I found that it is easier to run those in reverse order with the VBA macro saying to put the copied spreadsheet at the beginning. Even though SAS and Excel do not talk together very well, there are still ways to trick them into communicating! Counseling is not an option in this case.

Now, by running the code in SAS, we have opened Excel, had it read the files, and used macros to update all the graphs. Compare the following graph with our earlier template:

Figure 4. Final Excel Graph



Now, using the colors that have already been assigned in the template, the values have been updated, showing the date range that was requested. Assuming our template was correctly set up, nothing more needs to be done to this graph other than send it on to our audience. All of the desired functionality is already built in, and we have a presentation-ready graph produced in 10 seconds (half of which was waiting for Excel to open with our built-in 5 second timer). By using SAS X4ML programming and DDE, SAS effectively created a graph in Excel without any human factor.

TAKING IT A STEP FURTHER

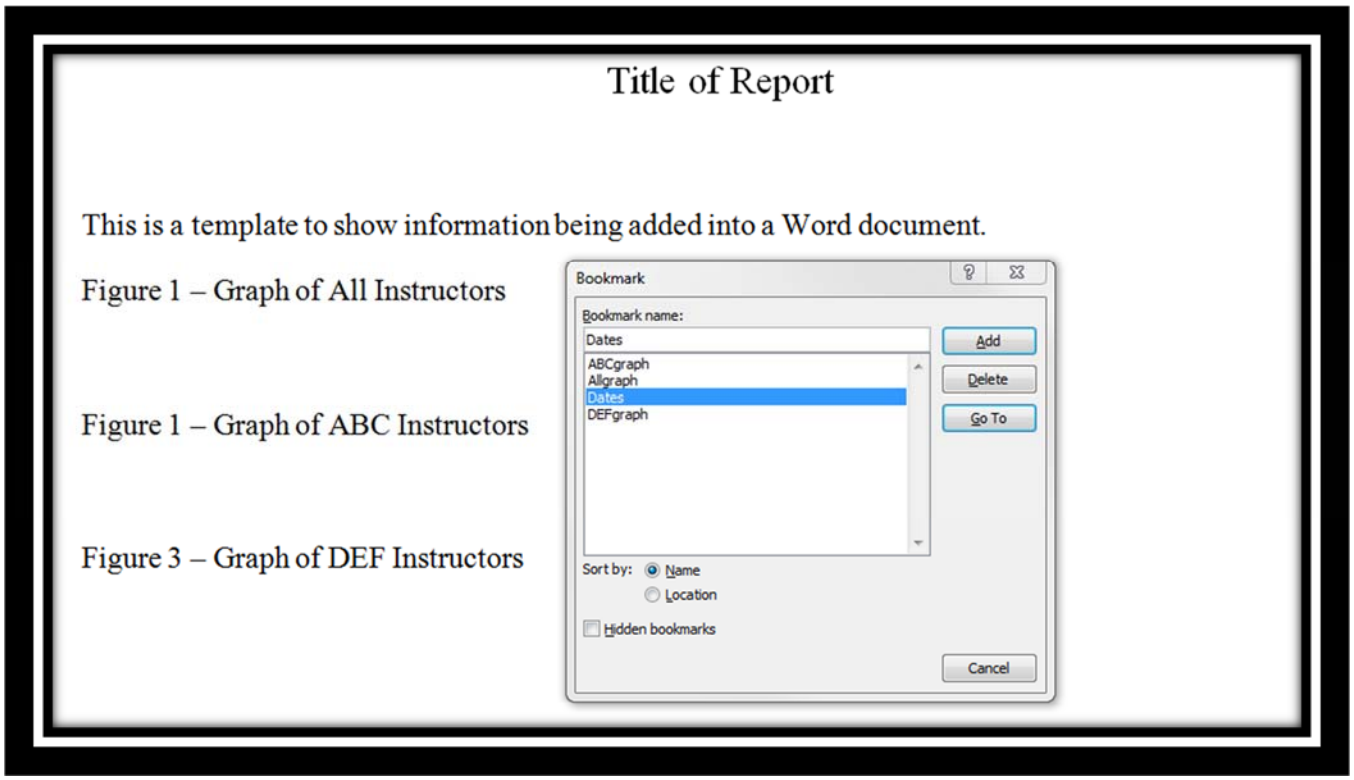
While this might be enough for one audience, it may not meet the needs of another. For example, what if we do not want the graphs in Excel, but to be put into a report that is created automatically in Word? Well, we could use SAS to put that in, but here, I again find that we can use VBA code to connect with Word. So, this is what we are doing:

Figure 5. SAS Control Process

SAS is calling up the code, Excel is taking that code and manipulating it, and Word is getting the finished result. While I will not be talking about VBA code for getting Excel to manipulate Word, please check out the globaliconnect site listed in the References to get detailed information about how this code works, as well as getting VBA macros that you can use in your own code.

Now, we have an Excel template, but for the finished product, I also want to create a Word template. In order to get Excel to find where to put specific information, I used bookmarks to mark where specific information should go. For example, in the title of the raw data set, I have a date range constructed. I then create a bookmark, called “Dates” here, and put that bookmark where I want dates to go. Then, I will use VBA to search for the “Dates” bookmark, then insert the applicable information there.

Figure 6. Word Template



VBA Code 5. Opening Word

```
Dim applWord As Object
Dim docWord As Object
Dim daterange As String

Set applWord = CreateObject("Word.Application")
applWord.Visible = True
Set docWord = applWord.documents.Open("C:\Users\wezupkoii\Documents\SAS Presentations\Word Template for Graph.docx", ReadOnly:=True)
```

Seen here, the Excel macro recognizes that it is now dealing with a word document and opens a read-only version, to prevent accidental mistakes with my template. Now, I need to place in the pertinent information. I have found that there are two ways to transfer text, by copying and pasting or by setting an Excel variable. I usually prefer to set an Excel variable, since there can be formatting issues when copying and pasting. For example,

VBA Code 6. Using Excel Variable to Place Word Information

```
daterange = Range("raw!A2").Value
docWord.Bookmarks("Dates").Range = daterange
```

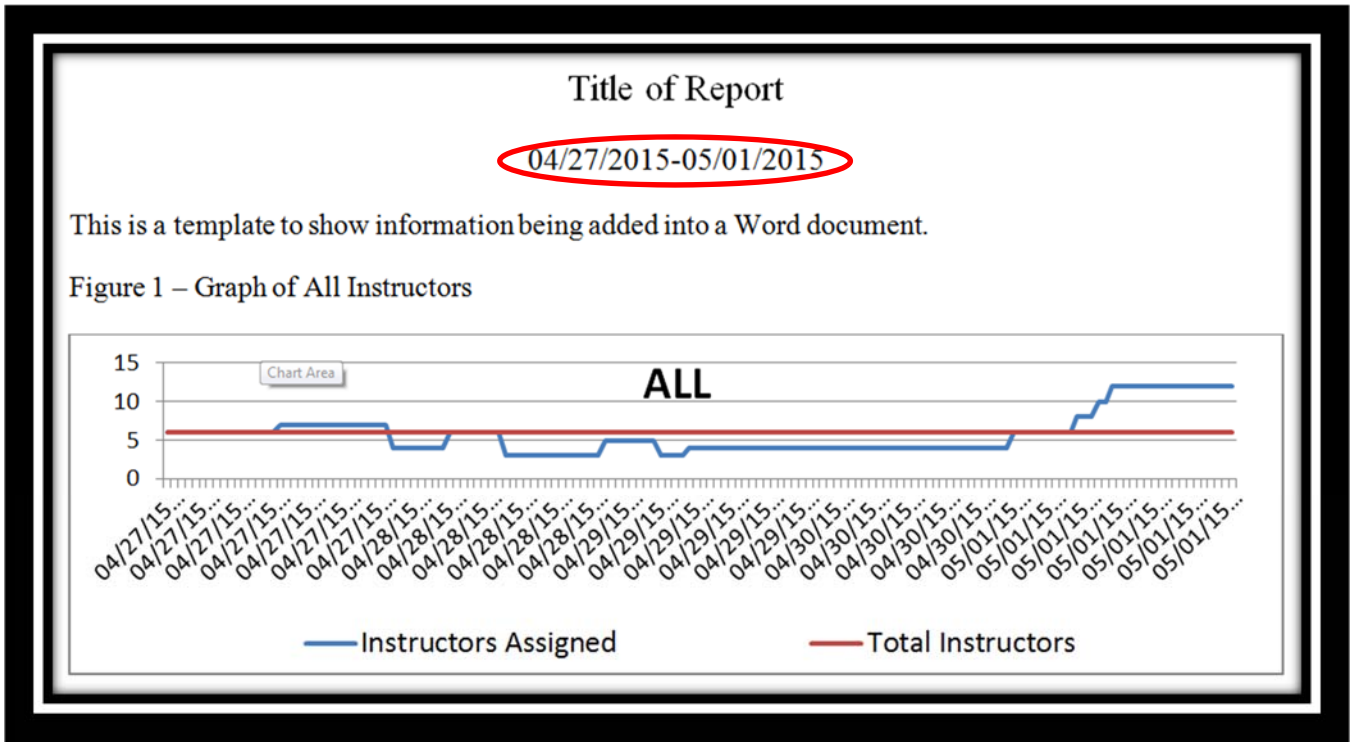
I have now placed the date range into my word document. Now that this is done, I also need to transfer my graphs. Since a graph cannot be an Excel variable, copying and pasting is my only option. I have created a bookmark called Allgraph for the first graph in Excel, which is the All graph, and placed it into the Word document here.

VBA code 7. Copying Graph

```
ActiveSheet.ChartObjects("Chart 1").Activate
ActiveChart.ChartArea.Copy
docWord.Bookmarks("Allgraph").Range.PasteSpecial
```

All of this code went into a new Excel macro, called Automate_Word. With the finishing touches of this macro, I am now able to call this macro in my file statement from before and see the finished product.

Figure 7. Word Template, Finished



The date range under the title went into the Word document perfectly formatted, needing no other work. While the graphs are getting into Word, formatting them still needs to be done. One way to lessen this is by making sure that the graphs are small enough to fit into a Word document before pasting them. However, then the Excel graphs might not be a useful size for my audience when using those graphs in Excel. Depending on wants and needs of the audience will say where it is better, to format before or format after.

This paper has covered a lot of information, especially since we have talked about three software packages (SAS, Microsoft Office, and DDE), how to communicate between them, and make presentation graphics ready in seconds. There is some upfront work that has to be done in order to enjoy these savings. For me and my audience, however, that work is paying rich dividends in the ability to have presentation ready graphics and reports available on call, instead of waiting hours or even days to prepare these materials.

CONCLUSION

SAS is an excellent statistical package, running the data efficiently and getting it formatted. When the audience prefers something different for whatever reason, SAS can also export the data into different formats. While this is also very useful, by using DDE, we can have SAS take direct control of Microsoft Office software and use it to further manipulate data or create graphs. Using the Record Macro function in Excel, one does not even need to be a VBA programmer to find out how to make changes in Excel and finish final formatting. By using the code in this paper, any VBA code can be modified to find custom data tags placed by SAS to update templates or create new graphs. Then, we can use VBA to send that data into any Microsoft Office software for desired results to be accomplished.

REFERENCES

Hawley, Dave. (2003, January 24). Re: Finding A Value and Returning Its Row Number [Online Discussion Group]. Retireved from http://www.ozgrid.com/forum/showthread.php?t=85615.

Globaliconnect. “Automate Microsoft Word from Excel, using VBA.” 2014. Available at http://www.globaliconnect.com/excel/index.php?option=com_content&view=article&id=169:automate-microsoft-word-from-excel-using-vba&catid=79&Itemid=475.

Lin, Choon-Chern. 2006. “Step-by-Step in Using SAS DDE to Create an Excel Graph Based on N Observations from a SAS Data Set.” *Proceedings of the SAS Global 2006 Conference*>. San Francisco, CA: SUGI 31 http://www2.sas.com/proceedings/sugi31/154-31.pdf.

ACKNOWLEDGMENTS

I would like to thank the Federal Law Enforcement Training Centers (FLETC) and SAS for the opportunity to write this paper.

This paper 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 and not necessarily those of FLETC.

CONTACT INFORMATION

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

William Zupko II
U.S. Department of Homeland Security, Federal Law Enforcement Training Centers
William.ZupkoII@fletc.dhs.gov