Paper 036-2012

Excelling with Excel

Tim Beese and Greg Granger, SAS Institute Inc., Cary, NC

ABSTRACT

Microsoft Excel is the most widely used tool for data analysis, and it is the default entry point for many consumers wanting to explore and analyze data. While Excel is suitable for basic analysis, it does not provide the powerful analytics available in SAS®. With the SAS® Add-In for Microsoft Office, users of Excel can seamlessly leverage the power of SAS analytics while providing secure access to data and IT resources previously unavailable through Excel.

In this paper, the use of SAS Add-In for Microsoft Office running under Microsoft Excel is demonstrated. This paper also discusses the use of cell values and ranges in Excel worksheets as input and output to SAS® Stored Processes.

WORKING WITH SAS - FOR THE UNINITIATED

The initiated should feel free to skip this section.

The SAS system is a large and powerful collection of tools designed to analyze data. It is based around the SAS programming language. Fully describing this language is well beyond this or any single paper. If you are unfamiliar with SAS, "trust us." In this paper, we will show simple programs and explain them well. The heritage of SAS is rooted in statistical analyses and views data as collections of "observations." A single observation contains related data. For example, a person's name, age, phone number, ZIP code, and eye color are grouped into a single observation for an individual. A collection of observations is called a "data set" or "table." SAS organizes these data sets by manipulating them in a "DATA step," (which acts as a template of how to treat an observation) or by using a "proc", short for *proc*edure. For example, , "PROC PRINT;" prints a data set.

SAS also uses the concept of "libraries" and "files,", which are often referred to as "librefs" and "filerefs" respectively. A library is a collection of files, usually data sets. A file is a collection of information that can be in a large range of formats.

When SAS code runs, the code generally produces some form of result and a SAS "log" file. The log file shows the flow of the program and shows any warnings or errors that occurred while SAS tried to run the code.

Here is a very simple SAS program:

```
libname _ALL_ List;
run;
```

In this program, the first line requests that all library names be listed. The second line requests that the program (up to that point) run. The code is actually run when the program is "submitted," (which means that you are asking SAS to process the given code). This submission process results in a log. The following display is an abridged version of the SAS log that is generated by the previous code:

```
1     libname _ALL_ LIST;
NOTE: Libref=   SASHELP
      Scope=    Kernel
      Levels=   1
        -Level 1-
      Engine=   V9
      Physical Name= C:\SASv9\en\sashelp
      Filename= C:\SASv9\en\sashelp
NOTE: Libref=   MAPS
      Scope=    Kernel
      Access=   READONLY
      Levels=   1
        -Level 1-
      Engine=   V9
      Access=   READONLY
      Physical Name= C:\SASv9\en\maps
      Filename= C:\SASv9\en\maps
NOTE: Libref=   SASUSER
      Scope=    Kernel
      Engine=   V9
      Physical Name= c:\atemp\sasuser
      Filename= c:\atemp\sasuser
NOTE: Libref=   WORK
      Scope=    Kernel
      Engine=   V9
      Access=   TEMP
      Physical Name= C:\...\SAS Temporary Files\_TD7280
      Filename= C:\...\SAS Temporary Files\_TD7280
2     run;
```

Log Output 1: Listing Library Names

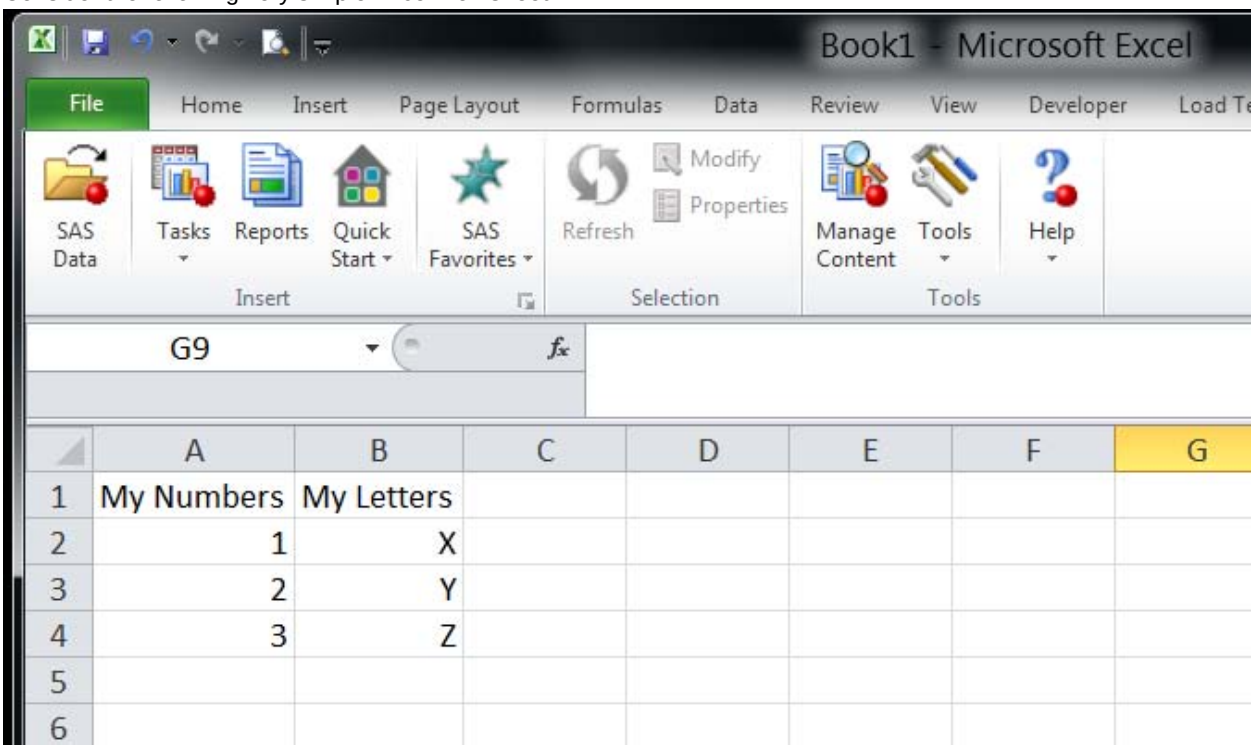
In the log, the numbers on the left are the line numbers for the log. The numbers generally match the line numbers in the original SAS program. You can see the log shows that the previous program contained four default library references: SASHELP, MAPS, SASUSER, and WORK. If you ran this code in SAS® Enterprise Guide® , the LIBNAME statement would not appear in the log until line 15 because SAS Enterprise Guide adds additional lines of code before and after any submitted code. If this program were run as a stored process (that you created in SAS Enterprise Guide), the LIBNAME statement would not appear in the log until line 30. Again, additional code is automatically added to the submitted code. In the original program if you replace LIBNAME with FILENAME, you get a list of files that were opened, or an empty list if no files have been opened.

WHAT IS A SAS STORED PROCESS?

SAS provides a way of packaging code along with the metadata that describes the code into an executable bundle called a SAS Stored Process. These stored processes enable you to execute SAS code by providing inputs and consuming outputs without worrying (too much) about the implementation process.

HOW DOES EXCEL SEND SAS DATA?

Consider the following very simple Excel worksheet:



Display 1. Sample Data in Excel

If you run a stored process that accepts range input and you specify the range A1:B4 for the input data source, the SAS Add-In for Microsoft Office would convert this range information into an XML document that looked like this:

```
<DATA>
	<EXCEL_TABLE>
		<Column1>My Numbers</Column1>
		<Column2>My Letters</Column2>
	</EXCEL_TABLE>
	<EXCEL_TABLE>
		<Column1>1</Column1>
		<Column2>X</Column2>
	</EXCEL_TABLE>
	<EXCEL_TABLE>
		<Column1>2</Column1>
		<Column2>Y</Column2>
	</EXCEL_TABLE>
	<EXCEL_TABLE>
		<Column1>3</Column1>
		<Column2>Z</Column2>
	</EXCEL_TABLE>
</DATA>
```

Each row is wrapped in an “EXCEL_TABLE” element, and each column is wrapped in a numbered “Column” element. This XML is then “streamed” to the SAS server.

WHAT IS A 'STREAM' (AND IS THERE A FIELD NEAR IT)?

A stream is just an ordered flow of data from a source to a destination. For example, the data would flow from the SAS Add-In for Microsoft Office to a SAS Server. For the previous XML, think of the stream as a "bucket brigade" that moves the characters from the client (Excel) to the server.

HOW DOES SAS USE A STREAM?

To SAS, a stream looks like any other file. As a simple instructional example, the following program echoes the information sent to it over the stream named 'instream'.

```
1    data foo;
2       infile instream length=lineLen ;
3       input @; * assign lineLen;
4       input lin $varying500. lineLen;
5       put lin;
6    run;
7
8    proc print data=foo noobs;
9    run;
```

The first line is a DATA step that creates a data set named 'foo.' The second line says use the contents of the 'instream' file as input, and when you read a line from this file, put the length of the line into the variable 'lineLen'. The third line requests a zero length input, which has the effect of setting the 'lineLen' variable (thereby telling us how long a line we should read). The fourth line requests the line be read into the variable 'lin' using a varying format (because the lines are of varying lengths) and accept only 'lineLen' number of characters. The fifth line outputs the contents of the 'lin' variable to the SAS log. The sixth line says run all of the previous the code statements before continuing to the next line in the program. The eighth line uses the PRINT procedure to print the data set 'foo.' (The NOOBS option says do not print the observation numbers on each line.) The ninth line requests that SAS run the eighth line before continuing.

OK, BUT I DO NOT LIKE XML. I JUST WANT TO START WITH A DATA SET!

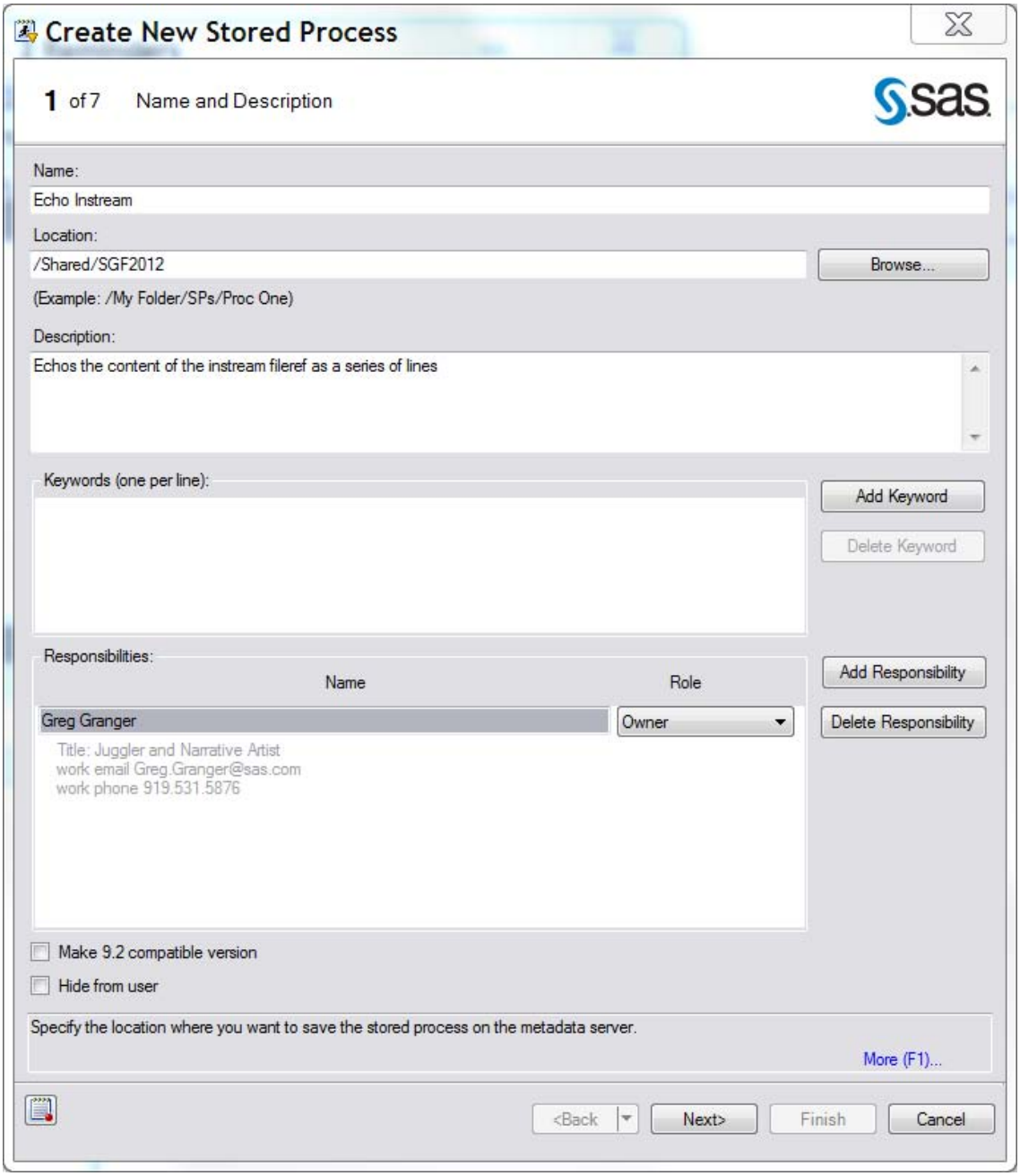
Absolutely! And using the LIBNAME engine you can. If you start your program with this line:

```
LIBNAME instream xml;
```

Your stream will look like a library. Remember how in the XML each row was wrapped in an "EXCEL_TABLE" element? When the LIBNAME engine examines the stream, it will put all these "EXCEL_TABLE" elements together and make them look like a library member. As a result, you can address your Excel data using 'instream.EXCEL_TABLE'. The SAS Add-In for Microsoft Office sets the macro variable '_WEBIN_SASNAME' to the value "EXCEL_TABLE." It is recommended that you use this macro variable to specify your data.

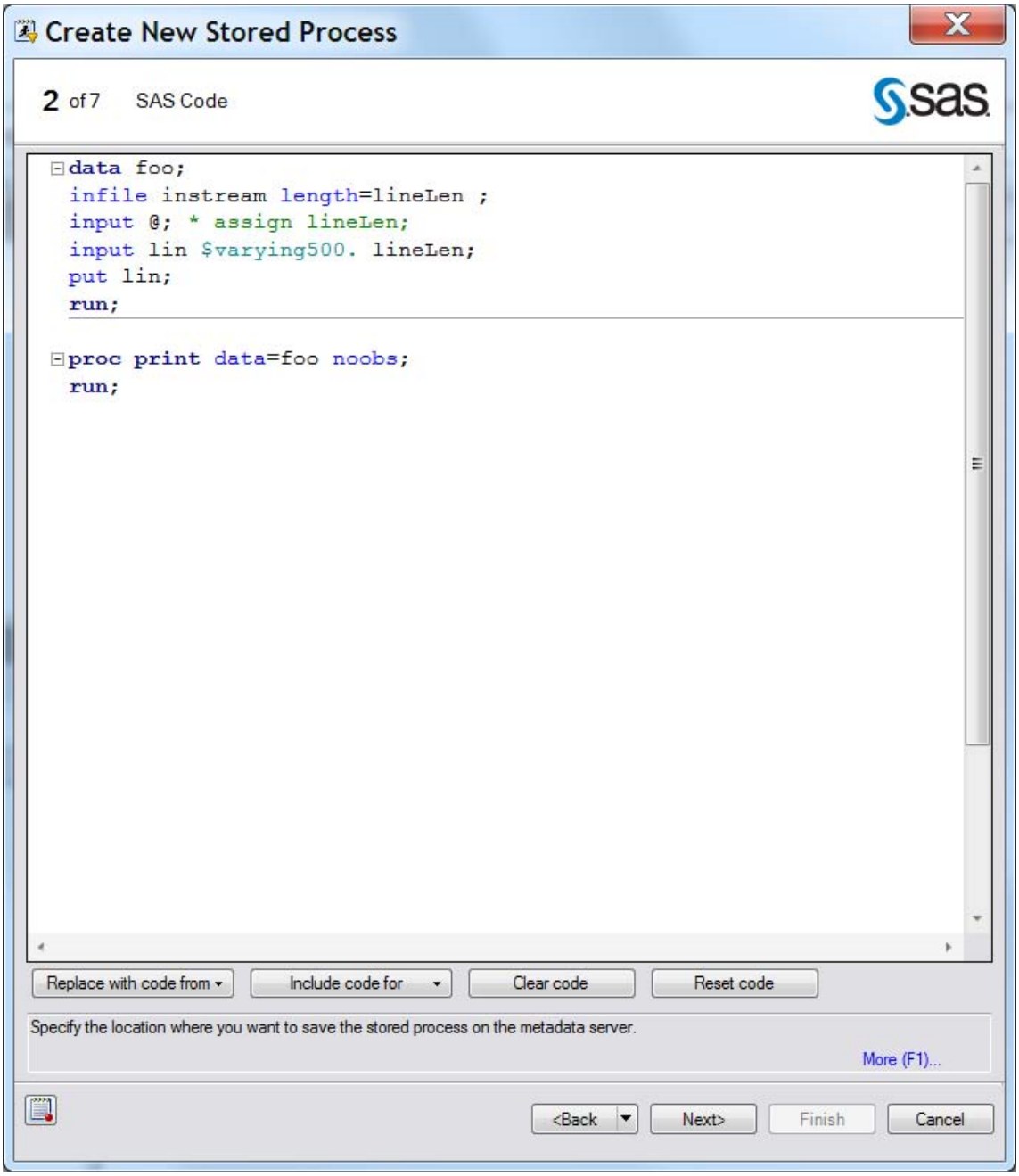
```
DATA EXCELDATA;
      SET instream.&_WEBIN_SASNAME;
```

CREATING A STORED PROCESS



Display 2. Create New Stored Process Wizard

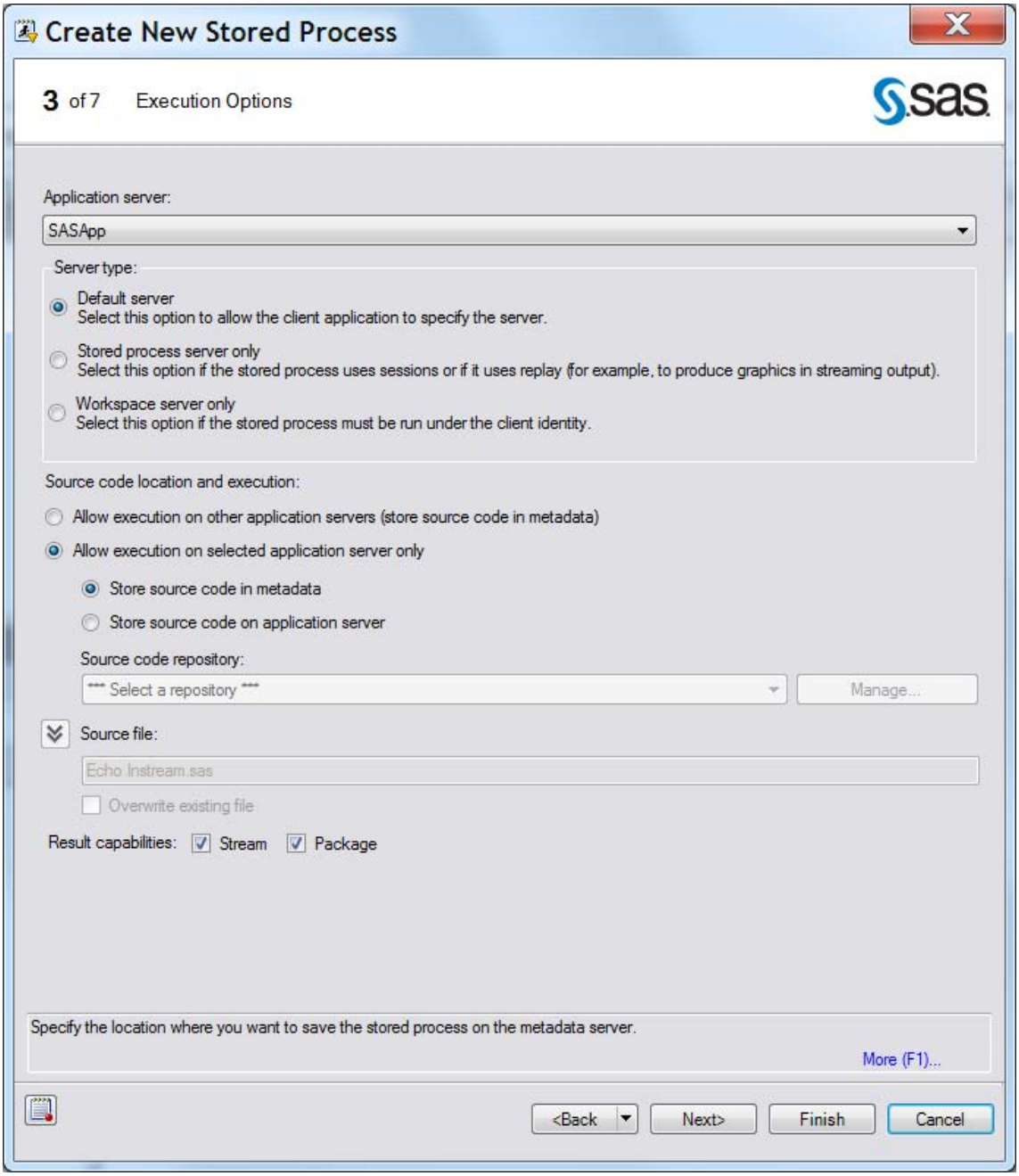
To start the Create New Stored Process wizard from SAS Enterprise Guide, select **File**->**New**->**Stored Process** . On the first page of the wizard, you define information (also referred to as the metadata) about the stored process , such as its name, location, description, associated keywords, responsibilities, compatibility, and visibility. Click **Next** to move to the next page of the wizard.



Display 3. SAS Code Page in the Create New Stored Process Wizard

The second page of the wizard shows the SAS source code, which is editable. If this had been a task-based stored process (a stored process created from the code of a SAS task), the source code would not be shown. This code is shown when you have a code node as part of your SAS Enterprise Guide process flow.

The Stored Process Wizard will only show seven pages if there is a disk based library reference that the code depends on. This information is given as page five. If no such dependency exists, then page five will be the "Data Sources and Targets" page and the wizard will only contain six pages total.

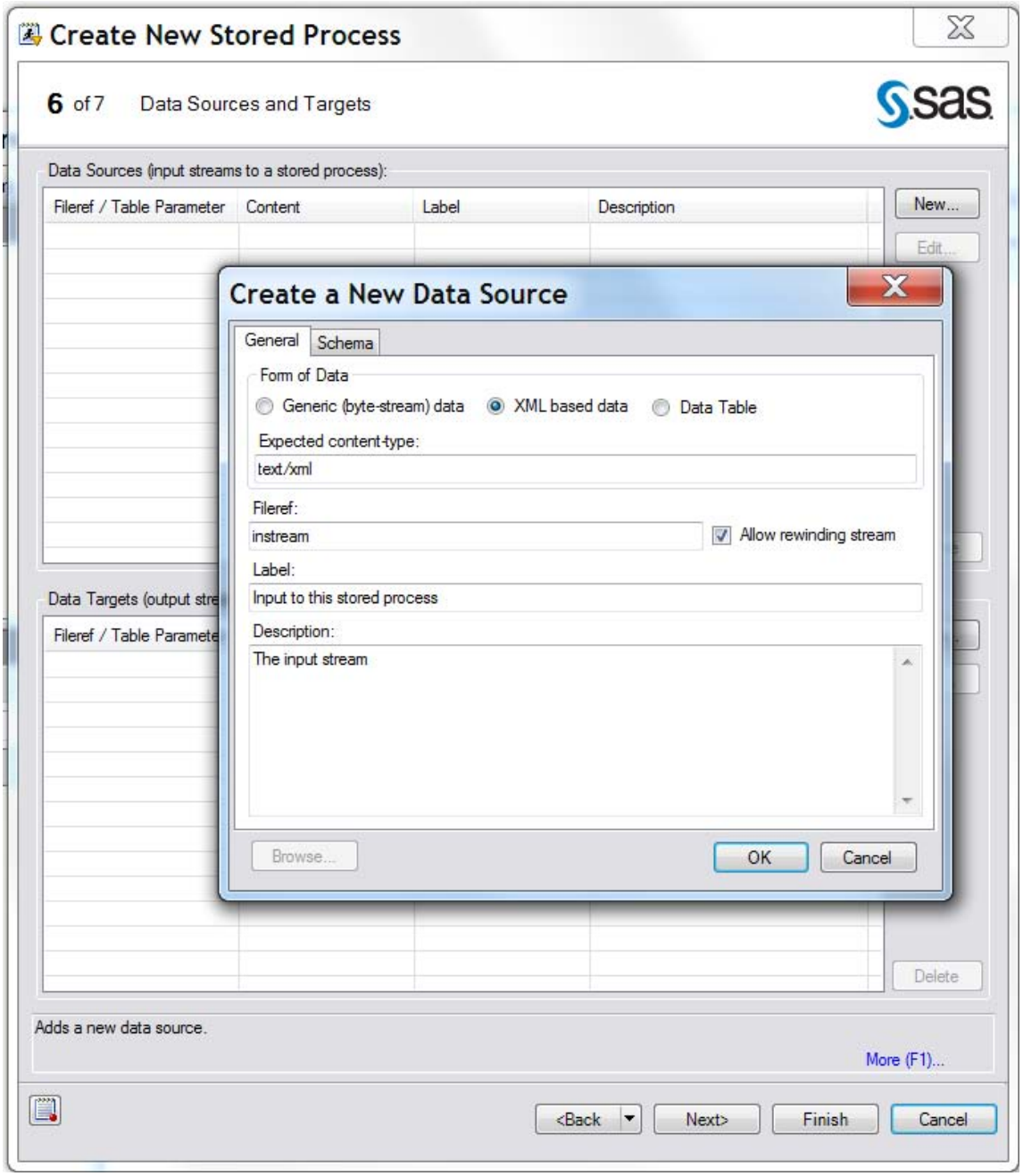


Display 4. Execution Options Page in the Create New Stored Process Wjzard

The third page of the wizard describes the name and type of SAS server to use to execute the stored process, the location of the code, and the capabilities of the results. To create a stored process that uses an input steam from Excel, the **Stream** check box must be checked.

For this stored process, the fourth and fifth pages are not needed, so we can skip them. We will come back to look at page five when we discuss prompts and output in the 'Defining an Input Prompt' section.

The sixth page of the wizard defines the input and output streams. As previously discussed, the SAS Add-In for Microsoft Office uses an input stream to communicate Excel information to SAS. Click **New** in the **Data Sources** area to open the Create a New Data Source dialog box.

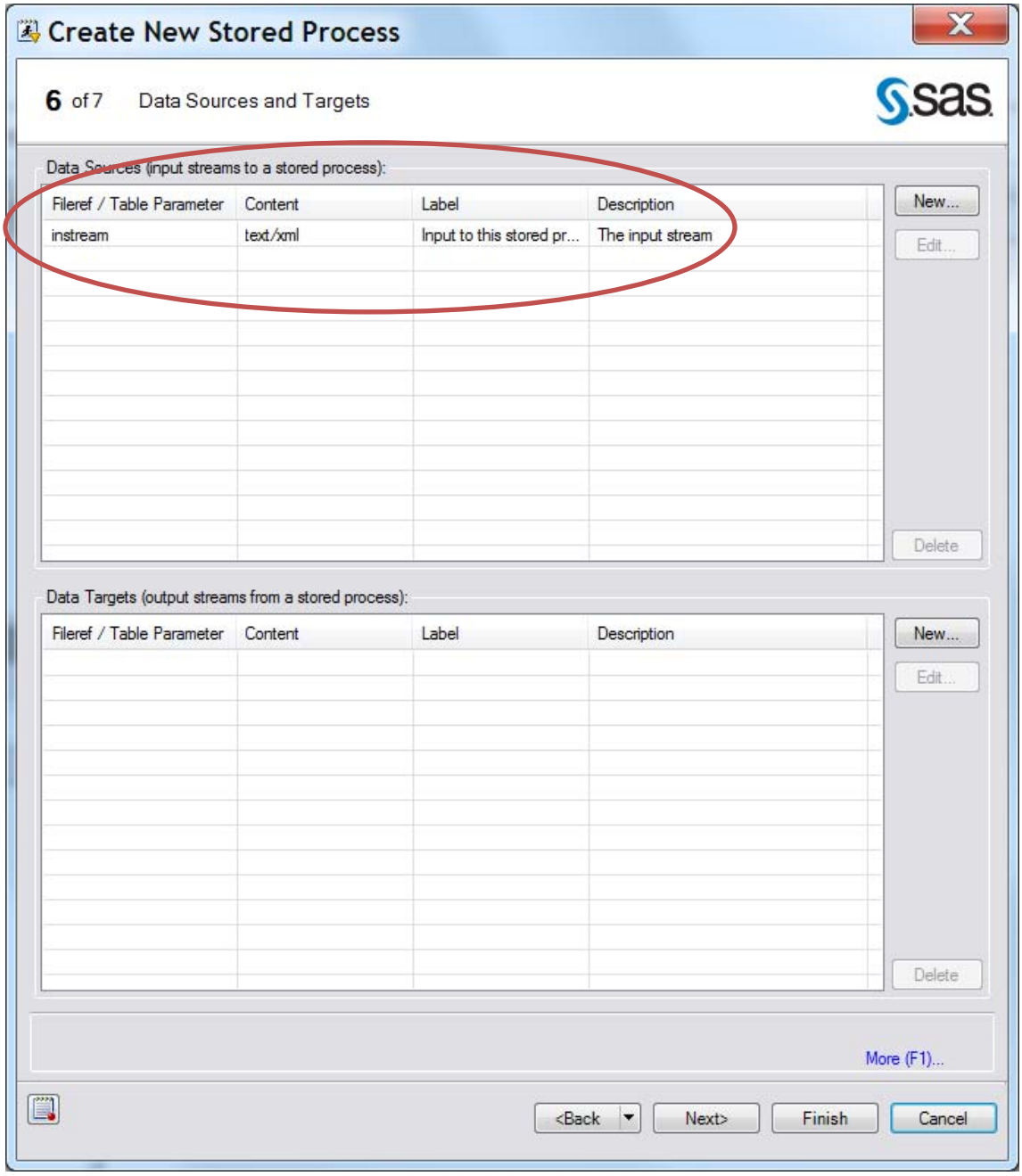


Display 5. Creating a New Data Source

The Create a New Data Source dialog box enables you to define and describe a data source. In this case, we are defining the 'instream' data source, which SAS will see as a file that SAS can reference under the name "instream." The **Form of Data** and **Expected content type** options tell SAS to expect XML data. Important to the functioning of the SAS LIBNAME engine is the **Allow rewinding stream** check box. This option must be checked when using the LIBNAME engine to dynamically interpret XML data. The LIBNAME engine must make multiple passes over the data, and these multiple passes can occur only if the stream allows its contents to be "rewound" (restarted from the beginning).

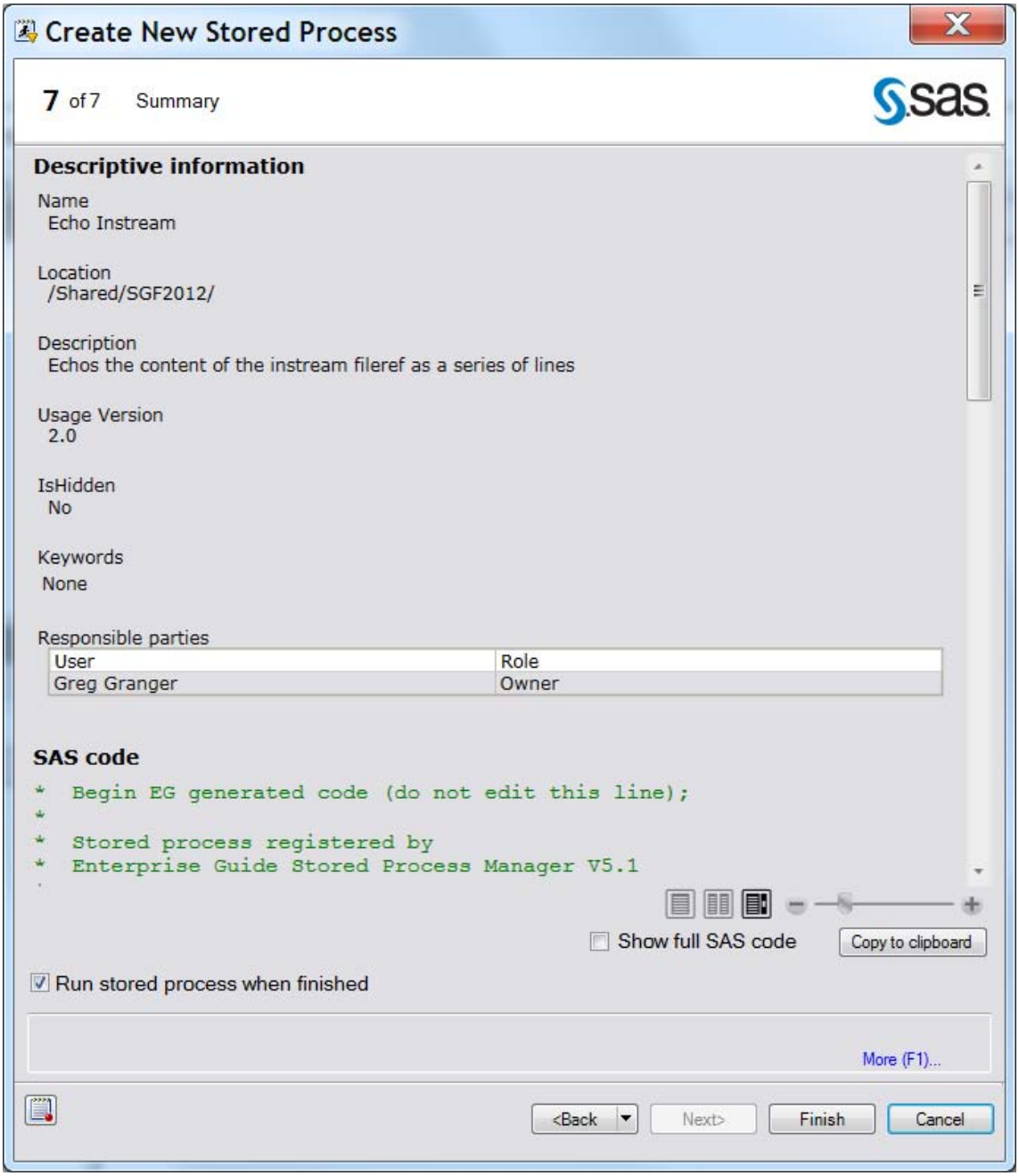
It is also very useful to provide a description of the data that you are expecting. When the user is prompted in Excel to provide the location of the input data, the user can see the description of the data source as a tooltip in the selection dialog box. If the stored process expects data that has columns with specific names, the user can get this

information from the tooltip and make sure that the data for the stored process meets the criteria needed for the stored process to execute correctly.



Display 6. Data Sources and Targets Page in the Create New Stored Process Wizard

After you click **OK** in the Create a New Data Source dialog box, page six is updated to show the instream data source (stream) .



Display 7. Summary Page in the Create New Stored Process Wizard

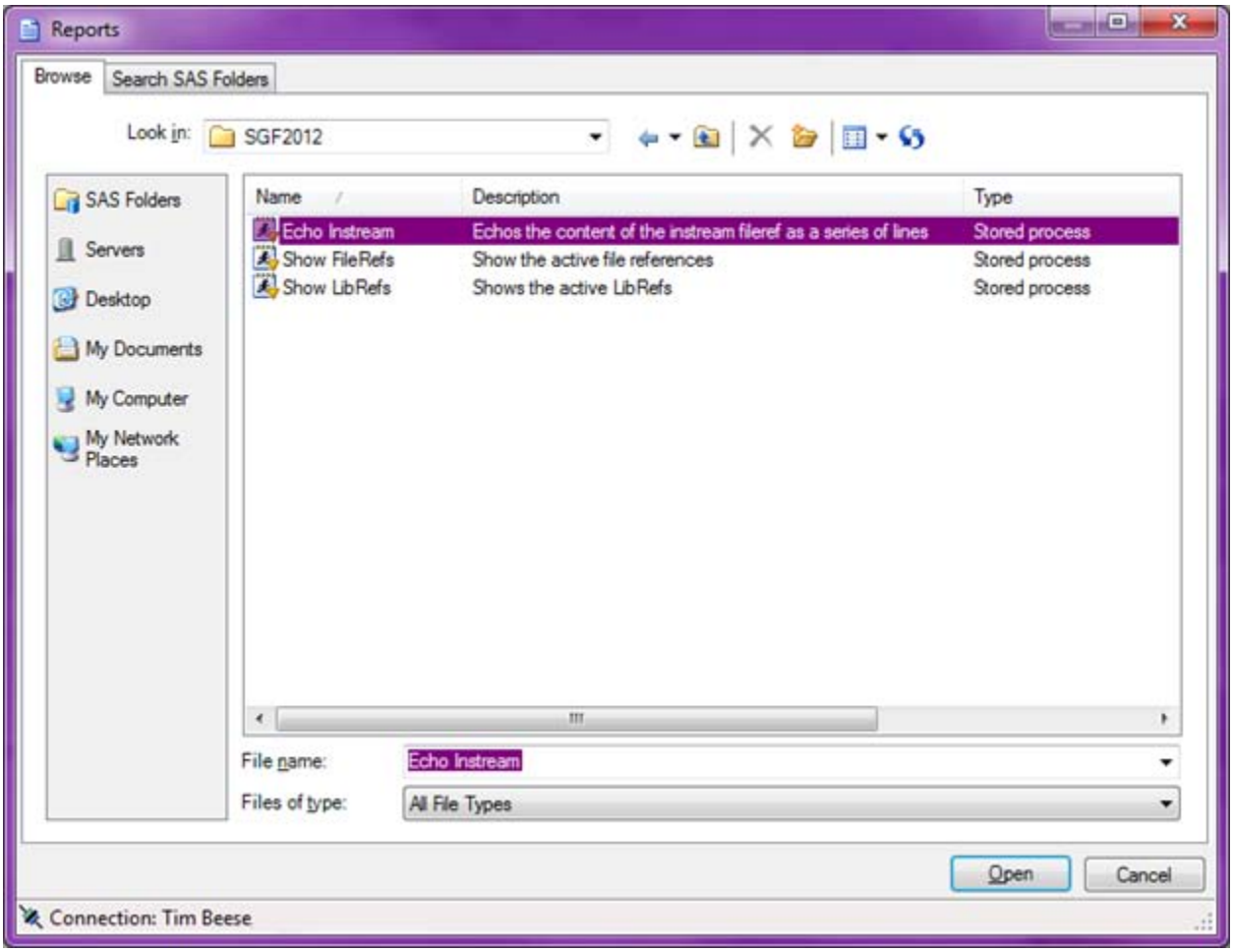
The last page of the stored process wizard is a summary page that lists the various choices you made when creating the stored process.

EXECUTING THE STORED PROCESS IN EXCEL

Now that you have created a stored process that takes an input stream and produces some output, it is time to run this stored process in Excel. Excel is the only Microsoft Office application where stored processes with input streams are supported. (This restriction is because Excel is the only application where it makes sense to store your data.)

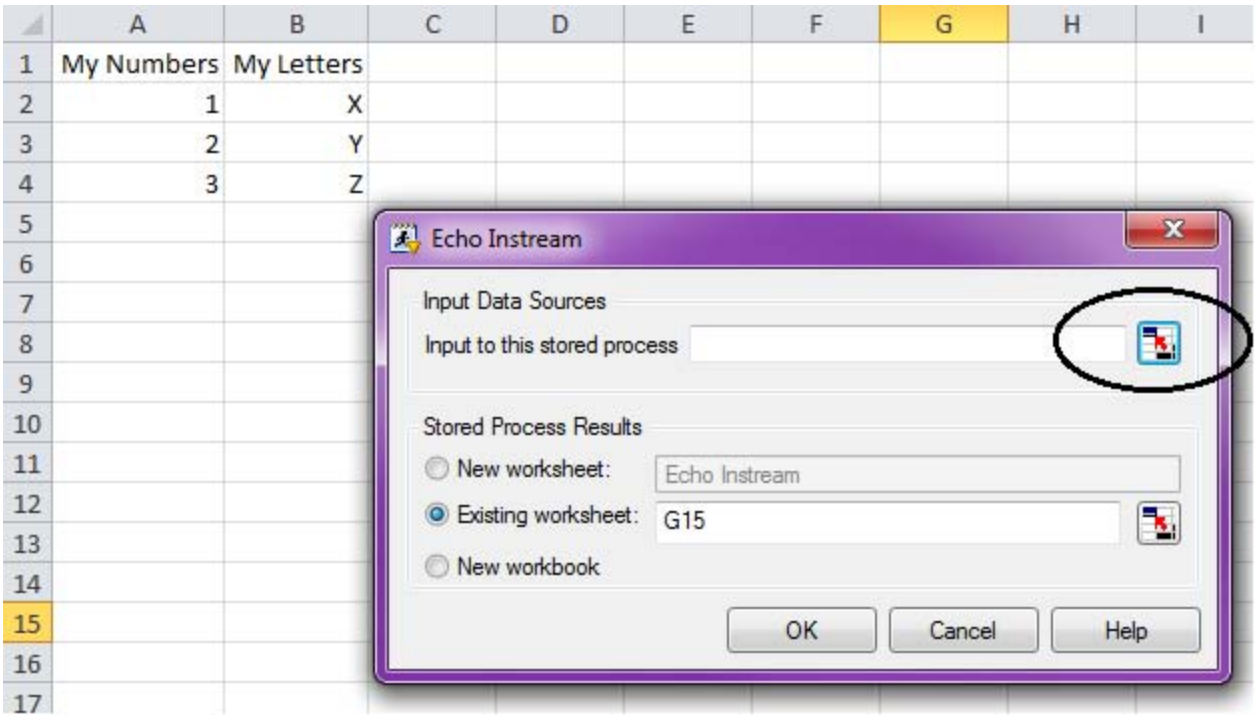
To run the stored process, open Excel and navigate to the **SAS** tab on the ribbon. The first group on this tab is the **Insert** group, which contains the options for inserting different types of content into your Office document. Stored

processes are considered a report, so click **Reports** to open a a dialog box that lets you choose the report to open. Stored processes are stored in SAS Folders. Navigate to the folder where you created your stored process and open it.



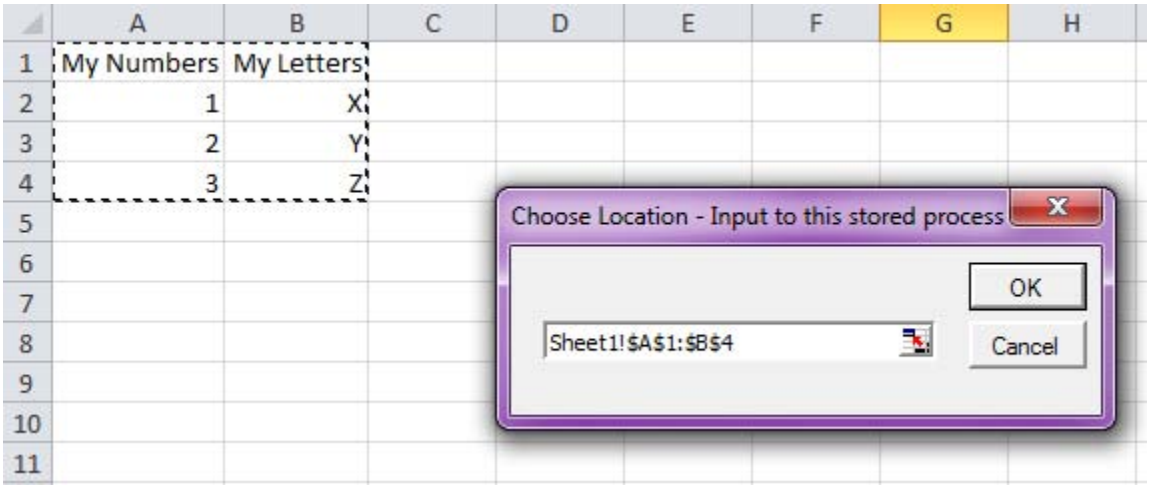
Display 8. Opening a Stored Process in Excel

In the previous section, we created a stored process called 'Echo Instream.' Now, we want to open that stored process into Excel. Before the stored process runs, the SAS Add-In for Microsoft Office examines the stored process and discovers that it requires an input data source. The SAS add-in prompts the user for the input data source.



Display 9. Using the Range Selector Button

From the dialog box for the input stream (in this example, the Echo Instream dialog box), you can use the range selector button to open the Choose Location dialog box. The Choose Location dialog box enables you to click in the Excel worksheet and choose a range of cells. The Choose Location dialog box will figure out the cell address for you.



Display 10. Using the Choose Location Dialog Box

If you do not have a range selector button in the Echo Instream dialog box, your security settings in Microsoft Office might be denying the SAS add-in access to the Visual Basic project. This access is needed to display the Choose Location dialog box. If the range selector button is not available in the dialog box for the input stream, there should be a link at the bottom of the dialog box that says 'Why am I unable to specify the location in Excel by clicking in a worksheet?' Click that link to open a help page that gives instructions for how to change this security setting.

After you have provided the range for the input stream, you can also choose where the SAS add-in will insert the results of the stored process. You can choose to put the results into a new worksheet and provide the name of that worksheet, or you can choose an existing worksheet. If you choose the existing worksheet, you can then use another

range selector button to select the cell where the SAS add-in will begin displaying the results. You can also type a cell address directly in the text box. Because we want to show these results on the same worksheet, we simply type F1, and the SAS add-in will use cell F1 on the active worksheet as the insertion point for these results.

When all of the locations have been selected, click **OK** to run the stored process. At this point, the stored process runs on the SAS server, and when the input stream is requested, the data from those cells will be sent upstream to the stored process and used in the execution. After the job has finished , the results will be sent back to the SAS add-in and displayed in the worksheet.

	A	B	C	D	E	F
1	My Numbers	My Letters				lin
2	1	X				<DATA>
3	2	Y				<EXCEL_TABLE>
4	3	Z				<My_Numbers>1</My_Numbers>
5						<My_Letters>X</My_Letters>
6						</EXCEL_TABLE>
7						<EXCEL_TABLE>
8						<My_Numbers>2</My_Numbers>
9						<My_Letters>Y</My_Letters>
10						</EXCEL_TABLE>
11						<EXCEL_TABLE>
12						<My_Numbers>3</My_Numbers>
13						<My_Letters>Z</My_Letters>
14						</EXCEL_TABLE>
15						</DATA>
16						

Display 11. XML Output in Excel

This stored process simply writes out the XML structure that is used by the XML LIBNAME engine to create a data set that represents the Excel data. The SAS Add-In for Microsoft Office tracks the data that was used as the input for this stored process, so if you were to refresh the stored process after the data had changed, the new cell values would automatically be used in the stored process.

For example, if we were to change the numbers and letters in the Excel data and then click **Refresh** in the **SAS** tab, the stored process would run again. This time, we would not be prompted (because the SAS add-in knows where the data came from), and the new values would appear in the results.

	A	B	C	D	E	F
1	My Numbers	My Letters				lin
2	8	A				<DATA>
3	9	B				<EXCEL_TABLE>
4	10	C				<My_Numbers>8</My_Numbers>
5						<My_Letters>A</My_Letters>
6						</EXCEL_TABLE>
7						<EXCEL_TABLE>
8						<My_Numbers>9</My_Numbers>
9						<My_Letters>B</My_Letters>
10						</EXCEL_TABLE>
11						<EXCEL_TABLE>
12						<My_Numbers>10</My_Numbers>
13						<My_Letters>C</My_Letters>
14						</EXCEL_TABLE>
15						</DATA>

Display 12. Changing the Input Data

MACRO VARIABLES AS PARAMETERS

SAS programs use a concept called “macro variables” to extend the SAS language. Macro variables enable SAS programs to symbolically name arbitrary blocks of text. In your SAS programs, these macro variables are placeholders that will be replaced at run time by their current text content. These variables can contain data or code, and are processed appropriately at run time. Stored processes use macro variables to provide input (known as a prompt) or produce output (known as an output parameter). The values of the input prompts can impact the execution your SAS code in limitless ways. When the execution of the program is complete, the state (value) of the macro variable can be returned as an output parameter.

A MORE COMPLEX EXAMPLE

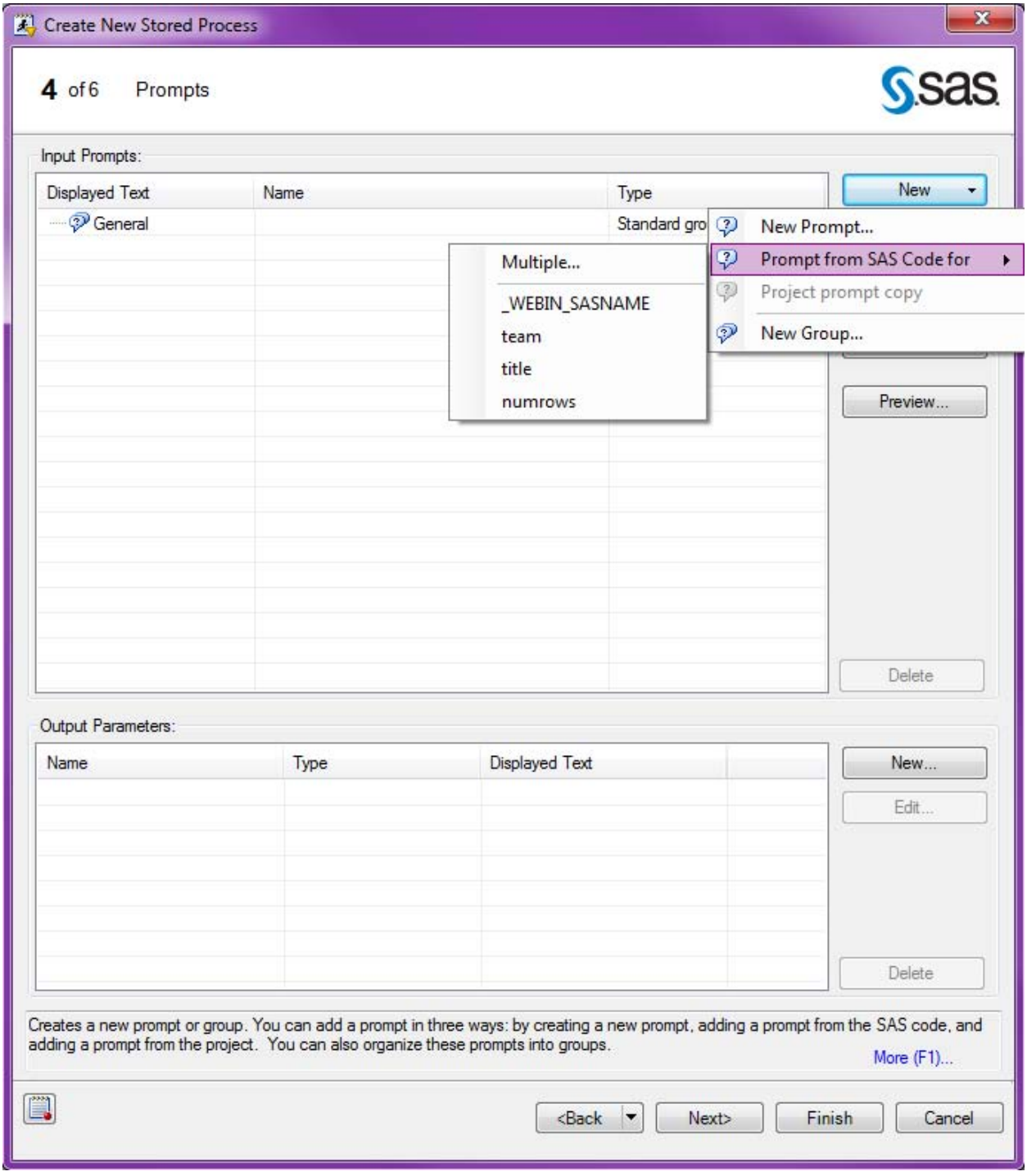
To demonstrate input streams, input prompts, and output parameters more thoroughly, here is a more complex example. The following code takes data describing NFL football players and their receiving statistics. It creates a report, ranking the players in various categories. Here is the SAS code:

```
/* options to put more information in the SAS Log */
 OPTIONS MLOGIC MPRINT;

/* Define an xml libname.  This is how the data is passed from Excel
into the stored process. */
LIBNAME instream xml;

/* Create a new data_set that will serve as the working input data source
throughout the stored process.  Create this data set by applying the filter
that might have been passed in as a prompt. */
DATA EXCELDATA;
       SET instream.&_WEBIN_SASNAME;
       WHERE TEAM = "&team";
RUN;

/* This macro will open a data_set and verify that it has at least one
observation.  If it has 0, then we will recreate the working input
data_set so that no filter is applied to the input stream data.
```

```
Basically, if the filter was not provided or an invalid filter was
provided, then we want to avoid using it. */
%macro validateDataFilter(ds);
	%let dsid = %sysfunc(open(&ds));
	%let nobs = %sysfunc(attrn(&dsid, NOBS));
	%let rc = %sysfunc(close(&dsid));

	%if &nobs = 0 %then %do;
		DATA EXCELDATA;
			SET instream.&_WEBIN_SASNAME;
		run;
	%end;
%mend;

/* Check to make sure that the working input data set exists and has at
least one observation.  If it does not, this macro will clear the filter */
%validateDataFilter(EXCELDATA);

/* Rank the receivers by receptions, yardage, and touchdowns.Create
three temporary data_sets that we will continue to work with. */
PROC RANK DATA=EXCELDATA
	DESCENDING
	TIES=LOW
	OUT=BASERANKS(LABEL="Top ranked receivers");
	VAR RECEPTIONS YARDS TD;
	RANKS RANK_REC RANK_YDS RANK_TDS;
RUN;

/* Take the final rankings and put them in order, using a simple formula that
adds all three rankings for each player and puts the player with the lowest
sum at the top.  This way the highest ranked players find their way to the
top of the list. */
PROC SQL;
	CREATE TABLE FINALRANKS AS
	SELECT PLAYER, RANK_REC, RANK_YDS, RANK_TDS
	FROM BASERANKS
	ORDER BY (RANK_REC + RANK_YDS + RANK_TDS);
RUN;

/* The only ODS output that we have from this stored process is a PROC print
of the table that shows all of the rankings.  */
TITLE;
TITLE1 "&title";
FOOTNOTE;
FOOTNOTE1 "Generated by the SAS System (&_SASSERVERNAME, &SYSSCPL) on
%TRIM(%QSYSFUNC(DATE(), NLDATE20.)) at %TRIM(%SYSFUNC(TIME(), TIMEAMPM12.))";

PROC PRINT DATA=FINALRANKS
	(OBS=&numrows)
	OBS="Rank"
	LABEL;
	VAR PLAYER RANK_REC RANK_TDS RANK_YDS;
RUN;

/* Finally, we need to assign the values of the output parameters.  To do this, we
want to find the player(s) that are ranked the highest in each category.  In case
there are ties, we want to separate them by commas. */
PROC SQL NOPRINT;
	SELECT PLAYER INTO :topRec separated by ", " FROM FINALRANKS WHERE RANK_REC=1;
	SELECT PLAYER INTO :topYds separated by ", " FROM FINALRANKS WHERE RANK_YDS=1;
	SELECT PLAYER INTO :topTds separated by ", " FROM FINALRANKS WHERE RANK_TDS=1;
RUN;
QUIT;
```

DEFINING AN INPUT PROMPT

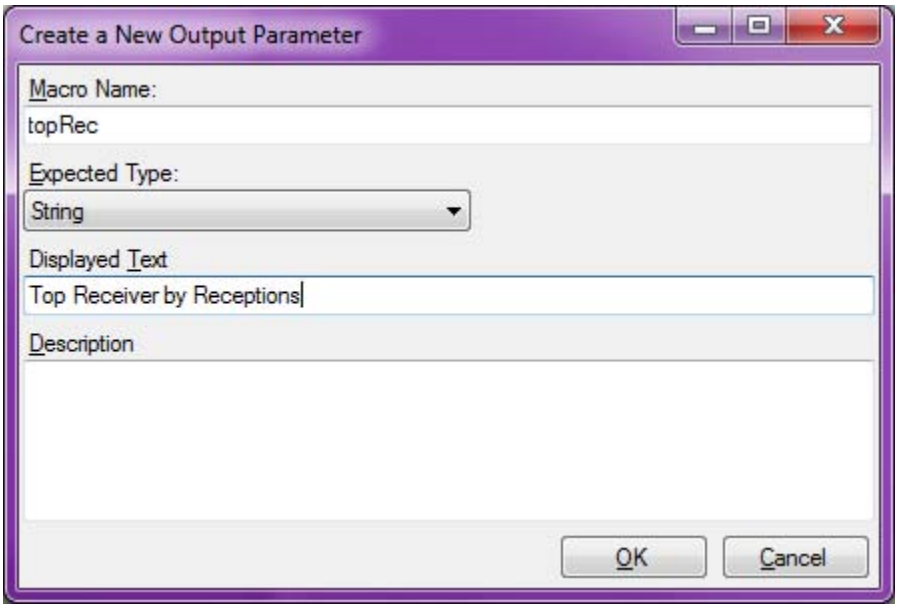
Input prompts can be easily defined. If you are working with existing SAS code, the Create New Stored Process Wizard will assist you by locating macro variables in the SAS code that can be used as possible prompts.



Display 13. Creating Input Prompts

Note in the previous screen shot that four different variables are offered as possible input prompts, including the previously discussed _WEBIN_SASNAME (which does not need to be defined as an input prompt because the SAS Add-In for Microsoft Office will automatically set its value). The other three prompts (team, title, and numrows) are all values that we want the user to provide.

After you select a variable to use or request that a new prompt be created, a dialog box will appear that lets you provide the name of the macro variable ("title" in this case) and a label to be used to describe the value for the macro variable in the **Displayed text** field. On the **Prompt Type and Values** tab, you can choose the type of prompt and the default value when they are presented with the prompts as they execute the stored process.



Display 14. Creating a New Output Parameter

New output parameters are even easier to create. The first entry is the name of the macro, followed by the expected type of the macro. Remember that all macro values are really just text. The expected type is a hint to the client application that this text might represent a different data type (such as numeric, date, and so on). As with input prompts, the **Displayed Text** and **Description** fields provide descriptive information that is used at run time to help identify the purpose of the returned value.

In this example, we will be setting the three output parameters to reflect the leader in each of the statistical categories.

PROVIDING PROMPTS FROM EXCEL

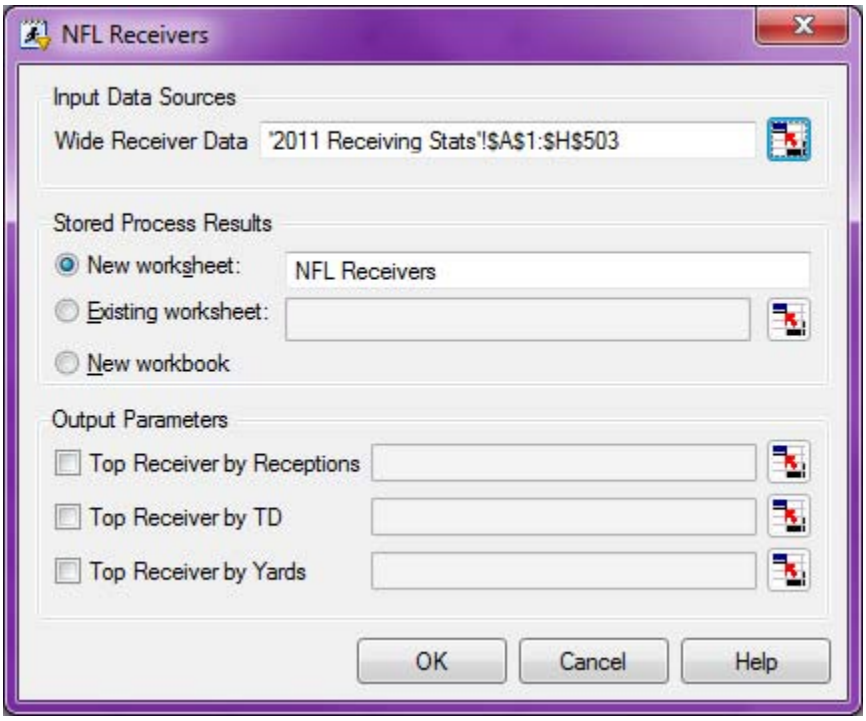
Now, you have a stored process that combines all of these features: input streams, prompts, and output parameters. It is time to put this stored process to work. First, you will need to have an Excel worksheet that contains your data. For the NFL Receivers stored process, we have a worksheet that contains receiving data for around 500 players in the National Football League (NFL).

To run this stored process in Excel, click **Reports** in the **SAS** tab and find the stored process that we just created. The first thing that we are prompted for is the prompts. The input prompts are shown in a separate dialog box since the prompts themselves are shared between several SAS applications.

The "Show top N" prompt lets us choose how many players to include in our final report. We also have the ability to filter so that we just have the prompts for a specific team. Finally, we can choose the name of the report. For the first run, we will use all of the default values.

Next, we are presented with a dialog box to select the input data source, the location for the stored process results, and the location of the output parameters. We have to provide the input data because the stored process cannot run without it. Using the range selector button, we can use the Choose Location dialog box to select our input data. If you hover over the input data source name or text box, the description of the data source will be shown as a tooltip. If the stored process author provided any description of the data, this tooltip might tell you what column names are expected with the input data.

To keep the results separate from the input data, we will put the results on a new worksheet called “NFL Receivers.” For the output parameters, we might not be ready to choose their output location. It is usually a good idea to see what the results look like first and then decide where the output parameters should go. We can leave the output parameters unchecked for our first pass. We can always access these later on.



Display 15. Supplying Input Data Sources from Excel

We run the stored process to see our results. The stored process runs on the stored process server and returns a top 10 list of receivers in a simple report.

	A	B	C	D	E
1			Top Receivers		
2					
3	Rank	PLAYER	Rank for Variable RECEPTIONS	Rank for Variable TD	Rank for Variable YARDS
4	1	Calvin Johnson	4	2	1
5	2	Wes Welker	1	6	2
6	3	Rob Gronkowski	5	1	6
7	4	Jimmy Graham	3	4	7
8	5	Victor Cruz	9	6	3
9	6	Roddy White	2	11	8
10	7	Larry Fitzgerald	12	11	4
11	8	Marques Colston	12	11	14
12	9	Steve Smith	15	21	5
13	10	Jordy Nelson	30	3	9
14					
15	Generated by the SAS System (SASApp, NET_SRV) on January 25, 2012 at 10:49:44 AM				
16					

Display 16. Sample Output in Excel

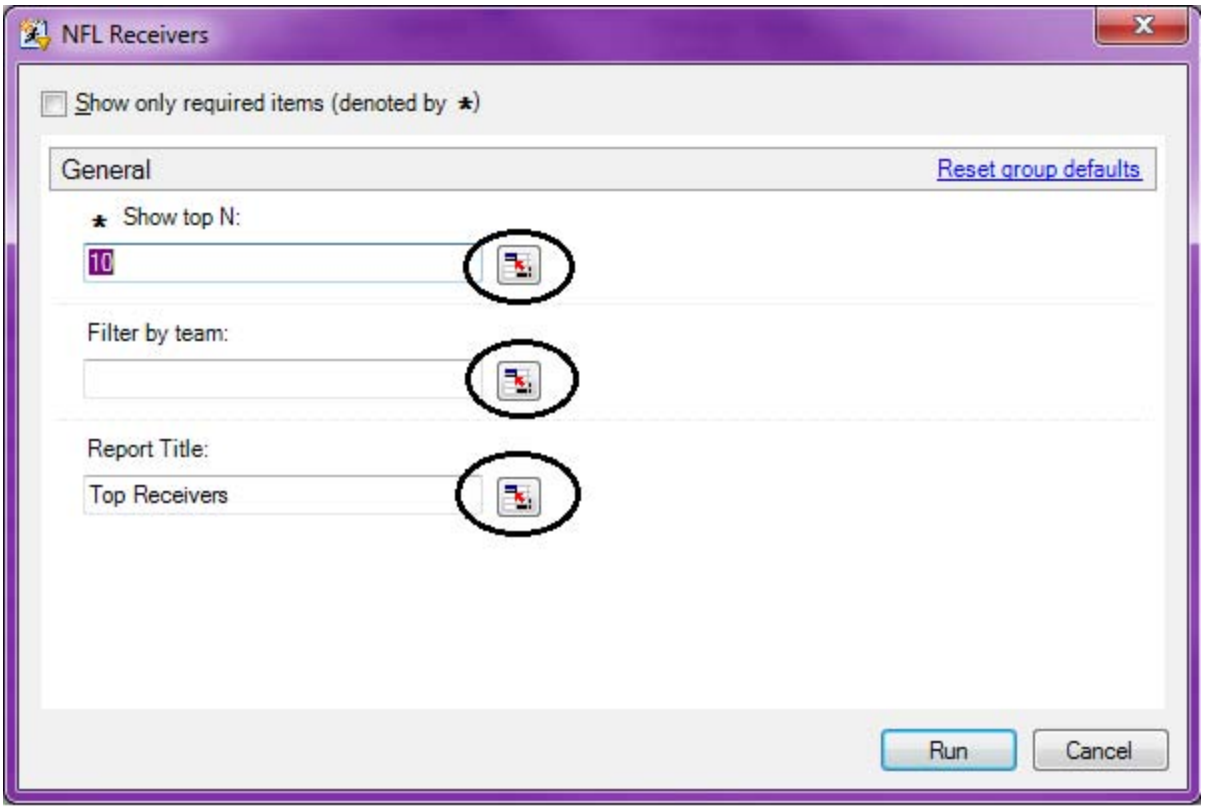
LINKING PROMPTS TO EXCEL CELLS

Now that we have the report output in Excel, we want to link the prompts to values in the Excel worksheet. We had three prompts before: one for the number of players to include, another for a team filter, and a third for the title of the report. We want to be able to change these prompts by simply modifying our worksheet. To do this, we need to add the values that we want to use for the report to our worksheet. We add those values to the side of the results.

	E	F	G	H	I	J
TD	Rank for Variable YARDS					
2	1			Show top N:	10	
6	2			Team Filter:		
1	6			Report Title:	Top 10 Receivers	
4	7					
6	3					
11	8					

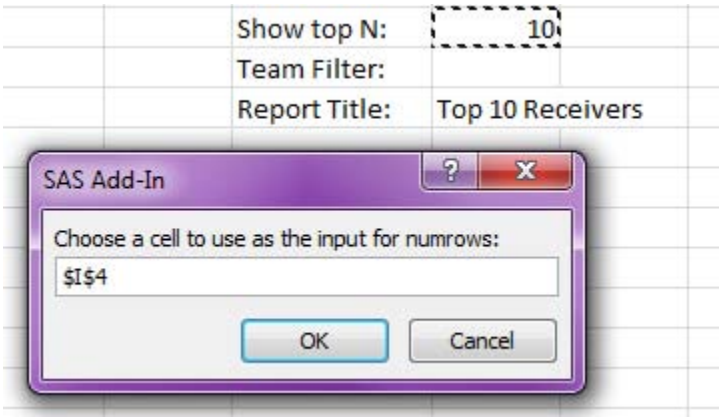
Display 17. Providing Prompt Values in the Worksheet

To link these values to the stored process, we need to run the stored process again. There are two ways to re-run a stored process. The first is to simply 'refresh' the stored process, which will run the stored process again without prompting the user. The same values are used for prompts and input streams as the previous run of the the stored process. The other way is to 'modify' the stored process, which will run the stored process again but show the prompts so that the user can change them. To wire up our cell values, we need to use the **Modify** option, which is in the **Selection** group on the **SAS** tab.



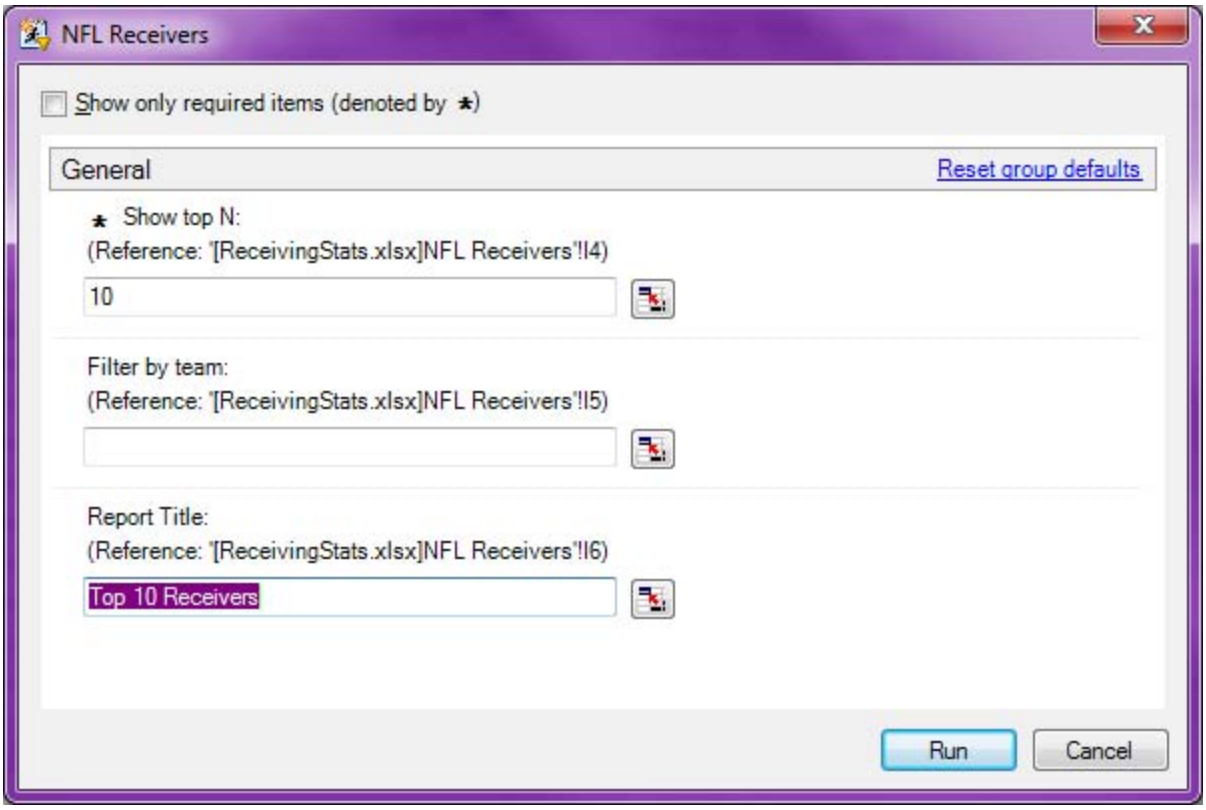
Display 18. Using the Range Selector Buttons

When we select **Modify**, the prompts dialog box appears again. Notice the range selector buttons on the prompts dialog box. These buttons will be available only in Excel and only for prompts that use simple text, numeric, or date types. To link a specific prompt value to a cell, click the range selector button and choose the cell in the Excel worksheet that contains the input value.



Display 19. Linking Input Prompts to Cell Values

The SAS Add-In dialog box tells you the name of the prompt that you are assigning, and after you select the cell with the input value, it displays the cell address. You can also type in the SAS Add-In dialog box to specify the cell address. After you click **OK** in the SAS Add-In dialog box, the prompt dialog box will be updated to show the cell address of the input data you selected. The SAS add-in will update the value in the prompt dialog box based on the contents of that cell each time that the dialog is shown.



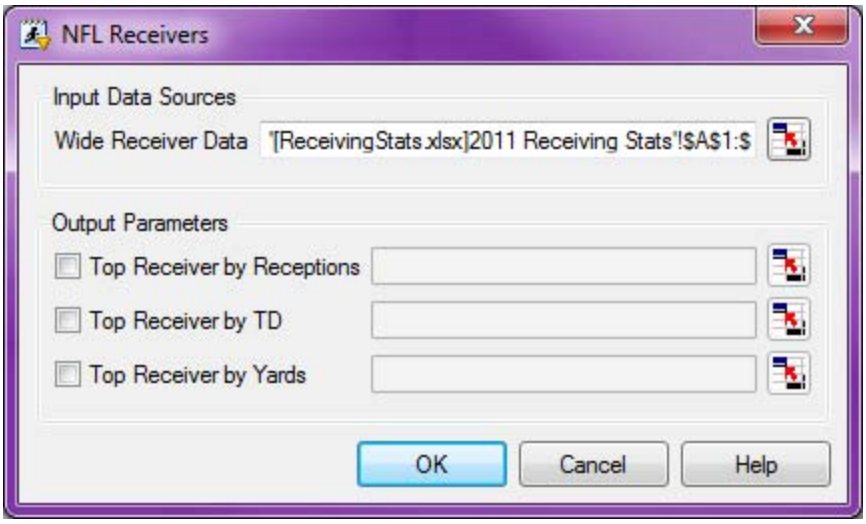
Display 20. Prompts Dialog Box with Linked Excel Values

Now in the prompts dialog box, you see the fully qualified cell address where the value is being retrieved , and the SAS Add-In for Microsoft Office will track that location. Even if you insert new cells or move cells around your worksheet, the SAS add-in will know where to find the prompt values the next time the stored process runs.

If you want to break the link with the cell, simply type the new cell address in the text box (to indicate that you want to provide a specific value), and the cell reference is removed. In the screen shot above, we have wired in all three prompts to their corresponding cells. Notice we still do not need to provide a value for the team filter if we do not want to, but we can still map to that cell so that if we provide a value later, it will be picked up. Now that our prompts are linked, click **Run** to continue.

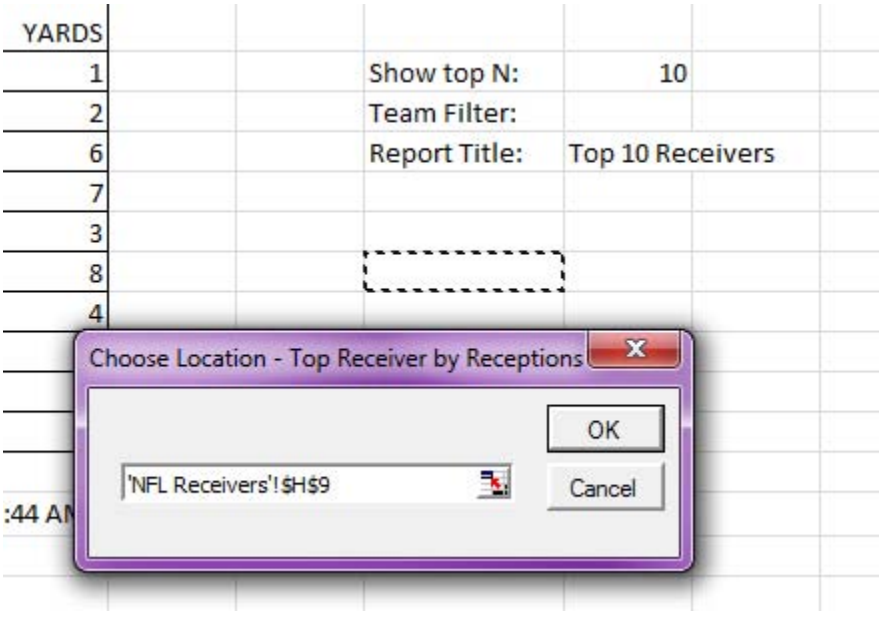
INCLUDING OUTPUT PARAMETERS

Now we are prompted with the input data and output parameter dialog box again. There is no need to re-specify the input data source because the SAS add-in remembers where the data was before and will reuse the same cells.



Display 21. Providing Output Parameters from Excel

Now we can insert our output parameters. We have the ability to control which output parameters we want to include in the report. We do not have to include any, or we can include specific ones. For each output parameter, we need to choose the cell location where they will be placed. We use the same mechanism with the range selector button to choose the location where each output parameter is displayed when the stored process runs.



Display 22. Choosing Excel Cells for Output Parameters

The title of the Choose Location dialog box tells you which output parameter you are choosing the location for. We will put the values of the output parameters in cells H9 through H11, choosing one cell for each output parameter. Now when we run the stored process, we will see all of our results in the worksheet.

	A	B	C	D	E	F	G	H	I	J
1				Top 10 Receivers						
2										
3	Rank	PLAYER	Rank for Variable RECEPTIONS	Rank for Variable TD	Rank for Variable YARDS					
4	1	Calvin Johnson	4	2	1			Show top N:	10	
5	2	Wes Welker	1	6	2			Team Filter:		
6	3	Rob Gronkowski	5	1	6			Report Title:	Top 10 Receivers	
7	4	Jimmy Graham	3	4	7					
8	5	Victor Cruz	9	6	3					
9	6	Roddy White	2	11	8			Wes Welker		
10	7	Larry Fitzgerald	12	11	4			Rob Gronkowski		
11	8	Marques Colston	12	11	14			Calvin Johnson		
12	9	Steve Smith	15	21	5					
13	10	Jordy Nelson	30	3	9					
14										
15	Generated by the SAS System (SASApp, NET_SRV) on January 25, 2012 at 11:06:40 AM									

Display 23. Sample Report with Output Parameters in Excel

We do not quite have the report that we want yet. Perhaps we want to add headings for the three output parameters so we know what they mean. We might also want to move those cells to the right so that they line up with our input prompts. If you select H9 through H11 and drag them over to the right, the cells are moved.The SAS add-in will recognize that the cells have moved, and when the stored process is refreshed, the SAS add-in will look in column I for where to put the output parameters.

Now we can add our headings and change some cell values for our input prompts. After we do this, since everything is linked to our worksheet, we can simply use the **Refresh** button on the **SAS** tab to run the stored process again, and we will not be prompted. The SAS add-in will look in the worksheet to the linked cells for the values to send to the stored process.

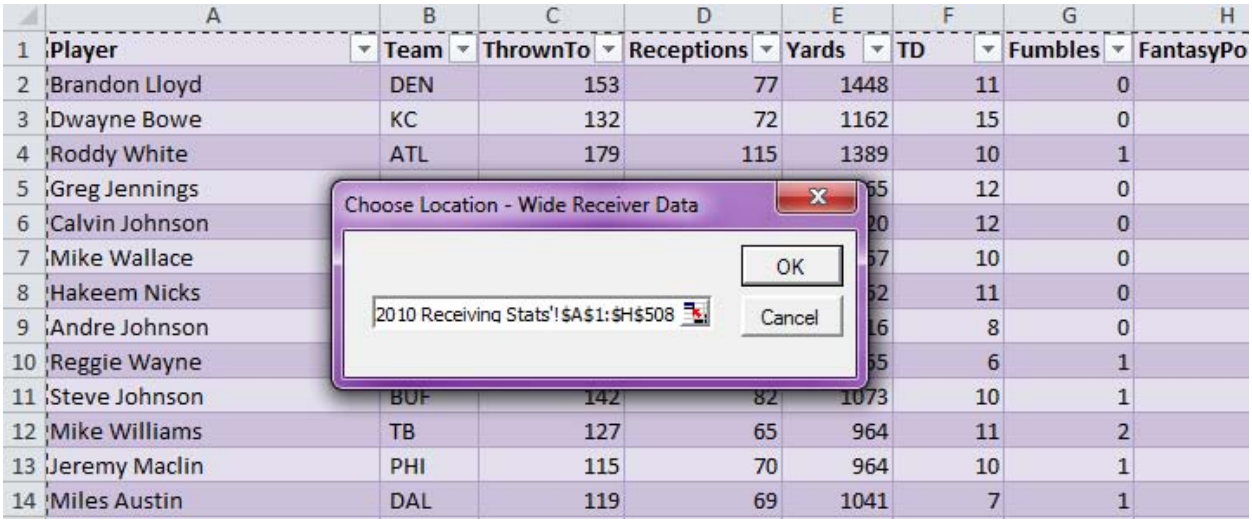
	A	B	C	D	E	F	G	H	I	J
1				Top 5 Dolphins						
2										
3	Rank	PLAYER	Rank for Variable RECEPTIONS	Rank for Variable TD	Rank for Variable YARDS					
4	1	Brandon Marshall	1	1	1			Show top N:	5	
5	2	Davone Bess	2	3	3			Team Filter:	MIA	
6	3	Brian Hartline	4	5	2			Report Title:	Top 5 Dolphins	
7	4	Anthony Fasano	5	2	4					
8	5	Reggie Bush	3	5	5					
9								Most Receptions:	Brandon Marshall	
10								Most TD:	Brandon Marshall	
11								Most Yards:	Brandon Marshall	
12										
13										
14										
15	Generated by the SAS System (SASApp, NET_SRV) on January 25, 2012 at 11:13:32 AM									

Display 24. Modified Report with Output Parameters in Excel

At this point, the stored process is fully interactive with the worksheet. You can simply update your worksheet with the values you want to see, and when you refresh the stored process, SAS will update your results.

CHANGING THE INPUT DATA

Another way that this becomes even more powerful is that when you modify the stored process, you can choose to change the input data source. The example we used was based off of 2011 data. Suppose you have other worksheets with 2010 and 2009 data. When you choose to modify the stored process, you can change the input data that you reference. In the Choose Location dialog box, you can select data on a different worksheet by simply navigating to that worksheet and choosing your input data. The new range will be tracked, and when the stored process runs, the SAS add-in will use the new range of data.



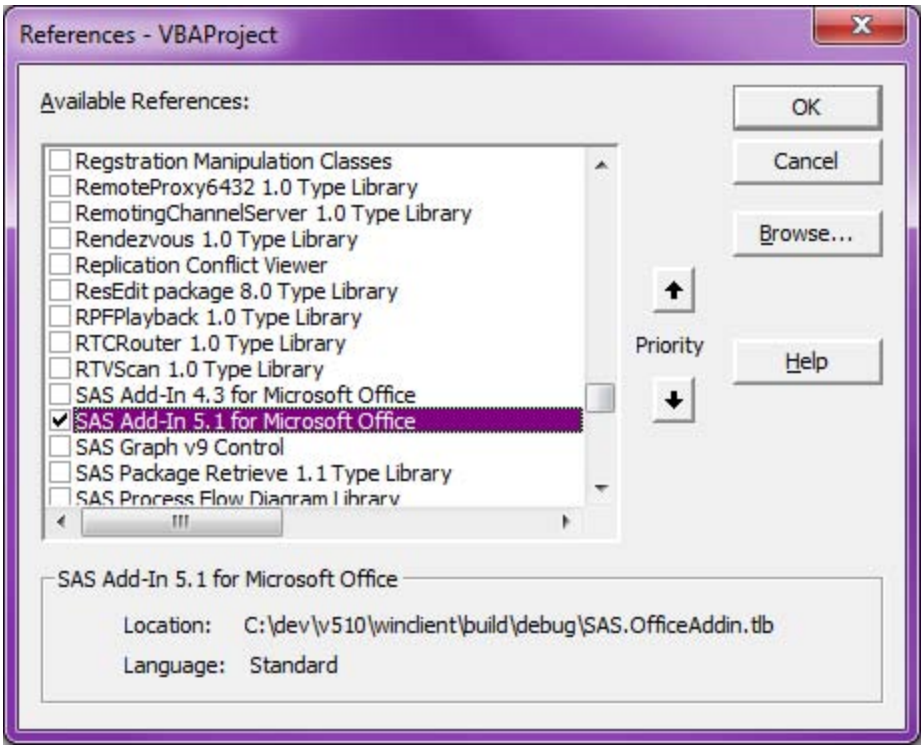
	A	B	C	D	E	F	G	H
1	Player	Team	ThrownTo	Receptions	Yards	TD	Fumbles	FantasyPo
2	Brandon Lloyd	DEN	153	77	1448	11	0	
3	Dwayne Bowe	KC	132	72	1162	15	0	
4	Roddy White	ATL	179	115	1389	10	1	
5	Greg Jennings				65	12	0	
6	Calvin Johnson				20	12	0	
7	Mike Wallace				67	10	0	
8	Hakeem Nicks				62	11	0	
9	Andre Johnson				16	8	0	
10	Reggie Wayne				65	6	1	
11	Steve Johnson	BUF	142	82	1073	10	1	
12	Mike Williams	TB	127	65	964	11	2	
13	Jeremy Maclin	PHI	115	70	964	10	1	
14	Miles Austin	DAL	119	69	1041	7	1	

Display 25. Changing the Input Data Source in Excel

REFRESHING FROM VISUAL BASIC FOR APPLICATIONS (VBA)

Using the scripting capabilities in the SAS Add-In for Microsoft Office, you can make this report even more interactive with the Excel worksheet. Instead of selecting **Refresh** from the **SAS** tab, you might want to have an Excel button in the worksheet that you can click that will refresh the result. Doing this is very easy with a little bit of VBA.

To do this, go to the **Developer** tab in Microsoft Excel and choose **Visual Basic** to access the VB Editor. Select **Insert**->**Module**, and a new code module is added to the workbook. You also need to add the SAS Add-In for Microsoft Office as a reference. To add this reference, select **Tools**->**References** and select the SAS Add-In 5.1 for Microsoft Office reference.



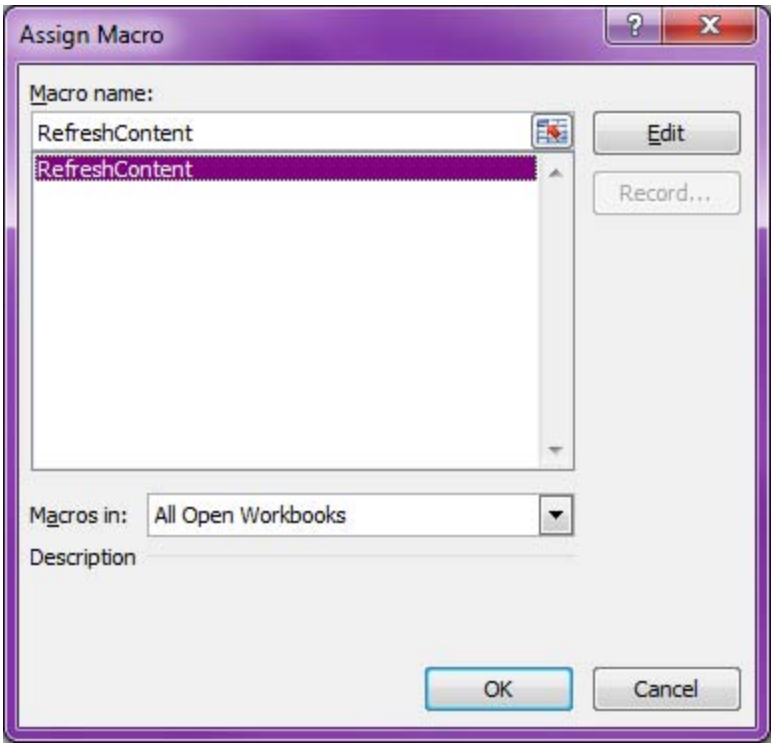
Display 26. Assigning a Reference to the SAS Add-In 5.1 for Microsoft Office

Using VBA to refresh the content will also work with the SAS Add-In 4.3 for Microsoft Office if you are using that release. After the reference to the SAS Add-In for Microsoft Office is added, you can write a macro that will refresh the content. Here is what that macro would look like:

```
Sub RefreshContent()

    Dim sas As SASExcelAddIn
    Set sas = Application.COMAddIns.Item("sas.exceladdin").Object
    sas.Refresh ("NFL_Receivers")

End Sub
```

To find out the name of the stored process to refresh on the "sas.Refresh" call, select the results you want to refresh and select **Properties** from the **SAS** tab. The Properties dialog box appears and shows the object name of the results. Specify the value of the object name in the Refresh method to make sure that this method refreshes the specified results.

After you have created the macro, the next step is to add a button in the Excel worksheet that calls it. On the **Developer** tab, select a button from the **Insert** drop-down list. Draw the button on your worksheet. When you are done you are prompted to assign a macro to the button. The RefreshContent macro that we just ran is available in the list, so if you select this macro, then you will have all of the wiring that you need.



Display 27. Assigning a macro to a button in Microsoft Excel

Now you can click the button in the worksheet to refresh the stored process. At this point, the user does not need to be aware that they are actually working with SAS. The user is simply interacting with the Excel worksheet, but leveraging the power of SAS through a stored process.

If you are interested in more VBA functionality, see "Tips and Tricks for Automating the SAS Add-In for Microsoft Office using Visual Basic for Applications." This paper was written for SAS Global Forum 2011 and shows many examples of scripting using the SAS add-in.

OTHER POSSIBILITIES - CHAIN STORED PROCESSES TOGETHER

Now that you have seen some examples of how to create stored processes and use input streams, prompts, and output parameters, you can begin to put together stored processes that blend the power of SAS with the environment of Excel and create a rich user experience.

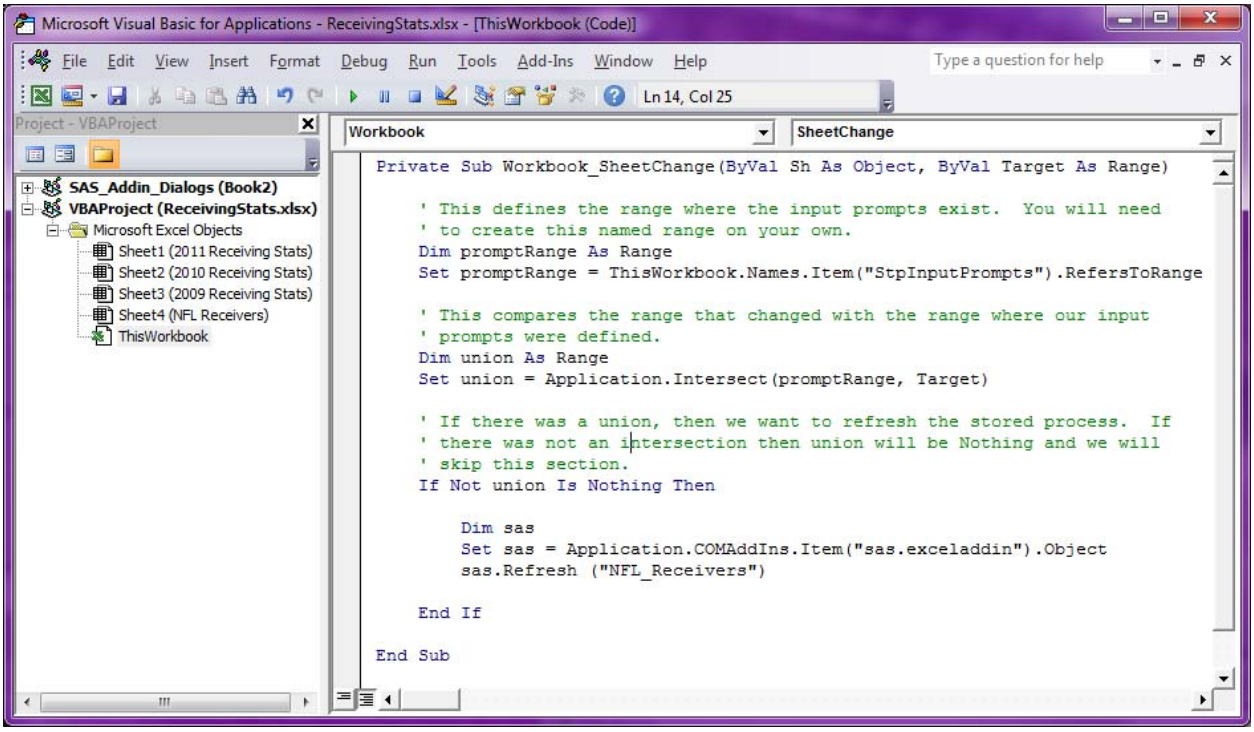
One possibility is chaining stored processes together. Chaining stored processes can be achieved by having the output of one stored process feed into another stored process. You can do this with an output parameter from the first stored process being written to a particular cell in Excel. Then, the next stored process can read that cell as the input value of a prompt . Now you have stored processes that chain together.

Using the SAS Add-In for Microsoft Office, you can control the order in which these stored processes are refreshed. In the Manage Content dialog box, you can use the up and down arrows to set the order of the stored processes, and then select the check box to refresh the analyses in order. Then it is as easy as choosing to refresh them all from the Manage Content dialog box. The result is an experience similar to a process flow.

OTHER POSSIBILITIES - AUTOMATING REFRESHES BASED ON EXCEL EVENTS

Another way that you can leverage Excel to use the power of SAS is to use VBA to listen for events in Excel. You could listen for a SheetChange event to tell you that the cell containing your prompt for a stored process was changed. When you detect that the cell changed, you could use the automation interface for the SAS add-in to force the stored process to refresh itself, which would pull in your new prompt values.

Doing this is quite simple. After you have your results and your prompts linked to cells, you can create a new named range that maps to the cells where your input prompts exist. Then, in the Visual Basic editor, find "ThisWorkbook" in your project tree and add the following code. The following example assumes that the name of the range that you defined is "StpInputPrompts," and the object name of your results is "NFL_Receivers".



Display 28. VBA to Automatically Refresh SAS Content

CONCLUSION

Microsoft Excel is a widely used application. Everyone is familiar with the environment, and many users store their data there. SAS is a very powerful application and can provide insight and knowledge about your data. Tying these two applications together provides a familiar environment with the accessibility of high-powered analytics. Stored processes and the SAS Add-In for Microsoft Office is the bridge to bring these two applications together.

The examples in this paper are just the tip of the iceberg. With the ability to get data from Excel and return data to Excel, stored process authors can write customized processes to deliver an Excel-like experience to their consumers.

REFERENCES

Beese, Tim. 2011. "Tips and Tricks for Automating the SAS Add-In for Microsoft Office using Visual Basic for Applications". *Proceedings of the SAS Global Forum 2011 Conference*, Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings11/012-2011.pdf

CONTACT INFORMATION

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

Tim Beese
SAS Campus Drive
SAS Institute Inc.

tim.beese@sas.com

Greg Granger
SAS Campus Drive
SAS Institute Inc.

greg.granger@sas.com