Paper SAS1831-2015

Teach Them to Fish—How to Use Tasks in SAS® Studio to Enable Co-Workers to Run Your Reports Themselves

Christie Corcoran and Amy Peters, Cary, NC, SAS Institute Inc.

ABSTRACT

How many times has this happened to you? You create a really helpful report and share it with others. It becomes popular and you find yourself running it over and over. Then they start asking, “But cannot you re-run it and just change ___?” (Fill in the blank with whatever “simple” request you can think of.) Don’t you want to just put the report out as a web page with some basic parameters that users can choose themselves and run when they want? Consider writing your own task in SAS® Studio! SAS Studio includes several predefined tasks, which are point-and-click user interfaces that guide the user through an analytical process. For example, tasks enable users to create a bar chart, run a correlation analysis, or rank data. When a user selects a task option, SAS® code is generated and run on the SAS server. Because of the flexibility of the task framework, you can make a copy of a predefined task and modify it or create your own. Tasks use the same common task model and the Velocity Template Language—no Java programming or ActionScript programming is required. Once you have the interface set up to generate the SAS code you need, you can publish the task for other SAS Studio users to use. Now that others can generate the output themselves, you actually might have time to go fishing!

INTRODUCTION

SAS Studio is a web application that enables you to access many features of the SAS System. It can be used to access your SAS data, libraries, and code. You can run existing code or use the rich Program Editor to modify code or write new code from scratch.

SAS Studio also contains snippets and tasks. Snippets are small units of SAS code you can paste into your current Program Editor. Tasks are yet another way to generate SAS code. This paper introduces you to SAS Studio tasks and steps you through writing your first one. You will be a pro in no time!

TASKS VS PROGRAMS

SAS Studio tasks are best described as fill-in-the-blank program files. As a first step, let us look at a basic SAS program.

A common statistical problem for students is to compute the combinations of N things taken K at a time. Dr. Goodnight wrote a SAS program that demonstrates a clear solution to the problem. For the rest of this paper, you will follow the steps that the SAS Studio team took to turn his program into a task.

Here is the SAS code to compute the answer:

```
/* Compute the combinations of n things taken k at a time */

%let n=4;
%let k=3;

data combo;
   array c[&k];
   n=&n;
   k=&k;
   if k>n then do;
      put "Error: k must be <= n";
      stop;
   end;
   keep c:;
                                 /* initialize */
   do i=1 to k;
      c[i]=i;
   end;
   output;                        /* output a combination */
                                  /* find next combination */
   do while(1);
      if c[k]<n then c[k]+1;      /* just increment last value */
      else do;
         n1=n;        /* move back down list until one is found */
         found=0;     /* to increment */
         do i=k-1 to 1 by -1 while(found=0);
            n1=n1-1;
            if c[i]<n1 then found=i;
         end;
         if (found=0)then goto term; /* none found ... finished */
         c[found]+1;
         do j=found+1 to k;
            c[j]=c[j-1]+1;
         end;
      end;
     output;                              /* output a combination */
   end;
term:;

proc print noobs;
run;
```

Notice that this program defines two macro variables, N and K. The macro variable N is the total number of objects. The macro variable K is the number of objects in a set. This SAS program creates an output data set of all possible combinations of these objects. Because the actual objects are unknown, numbers are assigned to uniquely identify them. The objects are numbered 1 to N. If there were four objects with three in a set, the output would be as follows:

1	2	3
1	2	4
1	3	4
2	3	4

Figure 1: Sample Output

If you want to change the number of the objects or the number of objects in a set, you need to modify the macro variables, N and K accordingly. If this SAS code was part of a SAS Studio task, these variables would appear as blanks in the task interface. From this interface, the consumer of the task could specify values for these variables.

WRITING YOUR FIRST TASK

You can take the program above, exactly as it is, and turn it into a task. To create a new task in SAS Studio, open the Tasks section in the navigation pane. Click and select New Task (as shown in Figure 2).

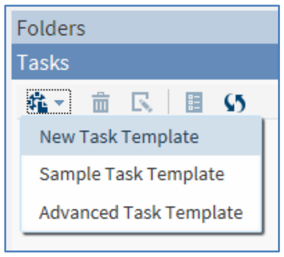


Figure 2: Creating a New Task

In SAS Studio, a new tab called New Task Template appears.

Tasks in SAS Studio are defined through Common Task Model (CTM) syntax. CTM is an XML definition. You will learn more about the basics of this syntax later. The only part you should look at now is the <CodeTemplate> declaration, which is the piece that generates the SAS code. Generation of the SAS code can be as easy or as complicated as you need it to be. Everything contained within this block of XML is written in the Apache Velocity Template Language. The power of the Velocity Template Language will help you get to wherever you need to go.

Notice that there is already some simple SAS code contained within this block of XML:

```
<CodeTemplate>
   <![CDATA[
proc print data=sashelp.cars;run;
   ]]>
</CodeTemplate>
```

In the code for the new task template, replace the PROC PRINT code with the Combinations code from above.

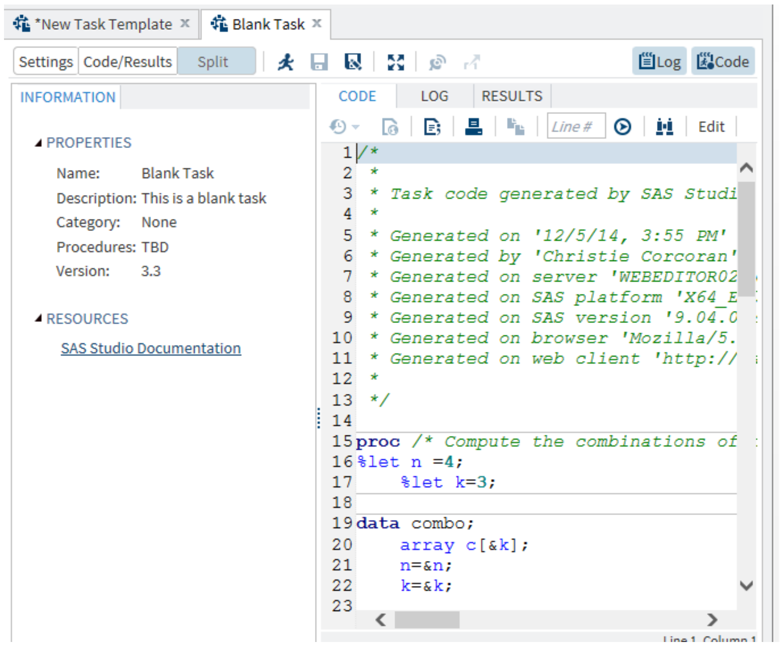
Click to run your task. The task opens in a new tab called Blank Task. There are two parts to the task. On the left is the task interface. On the right is the code produced by the task. See Figure 3.

Congratulations, you have just created your first task!

When you click in a task, SAS Studio submits the SAS code shown in the **Code** tab. You will notice that this task produces a table showing the calculated combinations as discussed in the last section.

To enhance your first task, you need to go back to the CTM. The **New Task Template** tab is still open, so you can switch back to it to continue working.

(TIP: To avoid confusion later, go ahead and close the **Blank Task** tab.)



Name: Blank Task
Description: This is a blank task
Category: None
Procedures: TBD
Version: 3.3

```
/*
 *
 * Task code generated by SAS Studi
 *
 * Generated on '12/5/14, 3:55 PM'
 * Generated by 'Christie Corcoran'
 * Generated on server 'WEBEDITOR02
 * Generated on SAS platform 'X64_E
 * Generated on SAS version '9.04.0
 * Generated on browser 'Mozilla/5.
 * Generated on web client 'http://
 *
 */

proc /* Compute the combinations of
%let n =4;
    %let k=3;

data combo;
    array c[&k];
    n=&n;
    k=&k;

```

Figure 3: Running Your First Task

You can also save your CTM file to work on later. Simply click 💾 to save the CTM file to a Folder or in My Tasks. You will notice that it is saved as a .ctm file.

READY, SET, GO! - ENHANCING YOUR TASK

Let's modify the task you had started earlier, but let's do it one step at a time so that you can see how each piece fits together. This example is just a small subset of what can be done. For more information about all of the pieces of the CTM, see *SAS Studio: Developer's Guide to Writing Custom Tasks.* (A link to this doc is provided in the **You Did It - What's Next** section of this document.)

THE COMMON TASK MODEL

The CTM consists of several sections that describe and define the task. A task consists of input controls that are then tied to the code that is ultimately produced. However, an input control's state could depend on the value of one or more input controls. In addition, the task could be considered in an invalid state should key requirements not be met.

The CTM is broken down into these sections:

Registration: This section provides information about the task. The task's name, description, and other identifying information are listed here.

Metadata: This section specifies the controls that will be used in the task. There are many controls for use in SAS Studio tasks. CTM has simple controls, such as check boxes and radio buttons, as well as more elaborate controls, such as the dual selector. The complete list of CTM controls can be found in our user documentation.

Note that the Metadata section does not lay out the user interface (UI) yet. The control definitions are split out from the UI for possible future expansion. It could be possible one day to have a different UI for different situations or devices.

UI: Using the controls defined in the metadata, the user interfaces to be presented to the end user are laid out. Note that for right now, only a top-down layout is supported.

Dependencies: This section defines any dependencies a control might have on other controls. For example, the state of a check box might enable or disable a text field.

Requirements: This section defines requirements that must be met in order for this task to be considered in a valid state. Many CTM controls already have a *required* attribute associated with them. This attribute indicates that the user must provide a value for the control before the task is valid. The Requirements section allows for more complex requirements to be defined. For example, a user must either fill in a text field or make a selection in a list box. If the user does neither, the task is in an invalid state.

Code Template: This section determines the output of the task (in other words, the generated SAS code). The code template is written using the Velocity Template Language.

MAKING IT YOUR OWN - THE REGISTRATION SECTION

The properties shown on the **Information** tab in Figure 3 come from the Registration section of your CTM file.

Scroll to the top of the code in the **New Task Template** tab to find the Registration section. Make the following two changes:

```
<Name>My Combinations Task</Name>
```

```
<Description>This is my first task!</Description>
```

Run the task. The **My Combinations Task** tab opens. Notice that the name of the tab now reflects the name you chose for within the Name tag. Also notice that both Name and Description properties on the **Information** tab have changed to match your changes.

ADDING AN INPUT CONTROL

Wouldn't it be nice to have the user of the task decide how many objects there are? Let's add an input control to your task.

The Common Task Model has many input types, such as check boxes, radio buttons, input fields, list controls, number steppers, sliders, and so on. All of these input types are documented in *SAS Studio: Developer's Guide to Writing Custom Tasks.*

Let's use a number stepper for our task.

Every task needs a container to put controls in, so first you will have to create a container. To accomplish this, you will first define both your container and number stepper control in the Metadata section. Then, you will update the UI section to add them to your task's user interface.

Close the **My Combinations Task** tab and go back to the **New Task Template** tab.

The Metadata Section

The metadata section of the CTM defines the input types that can be used in the task. It does not define the order in which the controls will appear. The CTM splits the definition of the controls apart from the layout of the task.

In your task, you will notice two tags already contained within the Metadata section: the DataSources tag and the Options tag. The DataSources section enables you to specify the type of input data for this task. The Options section enables you to specify the input controls for this task. In this section, you will define the container for the control and the number stepper for your task.

You will not be adding an input data source for this task, so you can ignore the DataSources tag. However, you will be adding code between the Options tags.

In the Options section, go ahead and add the following code for your number stepper control:

```
<Option name="nObjects" inputType="numstepper" defaultValue="4" minValue="1"
maxValue="100" >Total number of objects:</Option>
```

Because containers are used to lay out the UI, they are defined in the UI section. However, these containers need to be defined here in the Metadata section. In the Options section, add the following code to define a container called MY OPTIONS:

```
<Option inputType="string" name="optionsTab">MY OPTIONS</Option>
```

You now have defined everything that you need. You should have the following options defined in your metadata section:

```
<Options>

     <Option name="nObjects" inputType="numstepper" defaultValue="4"
minValue="1" maxValue="100" >Total number of objects:</Option>

     <Option inputType="string" name="optionsTab">MY OPTIONS</Option>

</Options>
```

Note that the order of these two options is not important. You are simply defining the metadata here.

Next, you need to add the control to the UI section, so the control appears in the UI when the task runs.

The UI Section

The UI section defines the layout of the task. Containers and Groups are two UI elements that help in arranging CTM controls. Currently, container elements are displayed as tabs in the task interface. Group elements appears as section titles on the tab.

All options must be within a container. If no container is specified, one will be created for you. Best practice, however, should be to always create one.

To create a container and to place your number stepper control in it, add the following code to the UI section:

```
<Container option="optionsTab">
     <OptionItem option="nObjects"/>
</Container>
```

Run the task. The **My Combinations Task** tab opens again. In the resulting task interface, you should see two tabs: MY OPTIONS and INFORMATION.

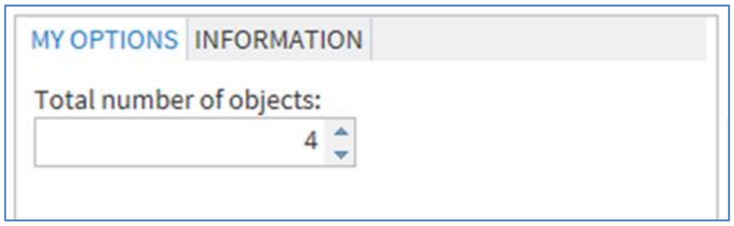


Figure 4: Using a Number-Stepper for Input

You are very close, but you aren't done yet! What you really want is for the number in the text box to be used in the generated SAS code. Let's do that now.

PUTTING IT ALL TOGETHER – TYING CONTROLS TO THE CODE TEMPLATE

Recall that SAS code generation is done through the Apache Velocity Template Language. Detailed documentation on Velocity can be found on the Apache website at http://velocity.apache.org.

It is easiest to think of Velocity as a fill-in-the-blank approach to generating the SAS code. The blanks are filled in by use of Velocity variables that are passed to the Velocity engine. It gets more complicated than that, but if you are not familiar with template languages, that is a good way to think about it for now. If you are already familiar with macro variables in SAS, then you are ahead of the game!

Close the task (the **My Combinations Task** tab). Go back to the **New Task Template** tab. The Code Template section should still contain the SAS code from the combinations program. If not, be sure to add it back in.

Look back at the name you gave the number stepper in the Metadata section. Notice that you named it nObjects. SAS Studio creates a Velocity variable that contains the value entered by the user. You can use that to find out how many objects the user wants.

In Velocity, variables are distinguished from other strings by adding a $ character to the beginning of the name. So in this example, the variable you are interested in is called $nObjects. Let's change the SAS code to use $nObjects instead of the macro variable N.

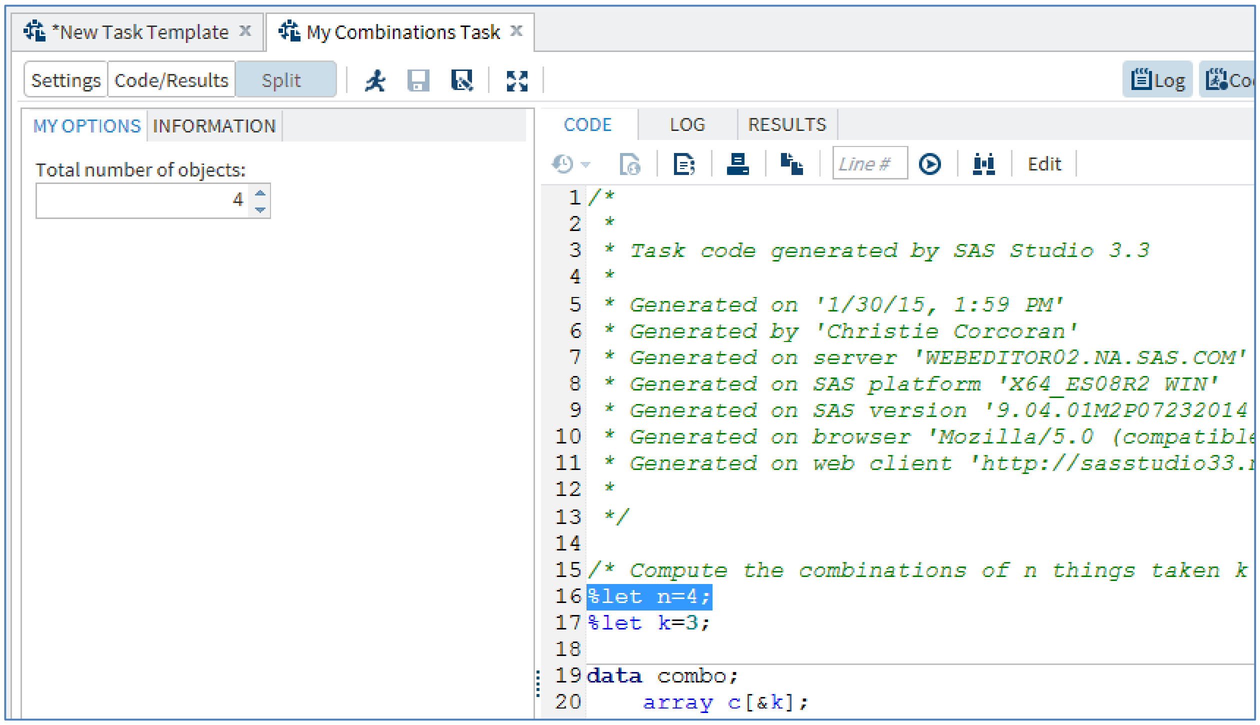
In the Code Template section, change the initialization of the macro variable from

```
%let n=4;
```

to

```
%let n=$nObjects;
```

Run the task. Use the up and down arrows to increase or decrease the value in the number stepper control. Notice that now the SAS code changes as you adjust the value of the number stepper control. Figure 5 is a screenshot of the resulting task interface and the SAS code. The SAS code associated with the number stepper control is highlighted.



```
/*
 *
 * Task code generated by SAS Studio 3.3
 *
 * Generated on '1/30/15, 1:59 PM'
 * Generated by 'Christie Corcoran'
 * Generated on server 'WEBEDITOR02.NA.SAS.COM'
 * Generated on SAS platform 'X64_ES08R2 WIN'
 * Generated on SAS version '9.04.01M2P07232014
 * Generated on browser 'Mozilla/5.0 (compatible
 * Generated on web client 'http://sasstudio33.
 *
 */

/* Compute the combinations of n things taken k
%let n=4;
%let k=3;

data combo;
    array c[&k];
```

Figure 5: SAS Code Affected by Number Stepper

ADDING ANOTHER INPUT CONTROL

As an exercise, see if you can add another number stepper control that enables the user to adjust how many objects are in a set.

After you try this exercise, compare your CTM with this final CTM. (Modified lines are highlighted in yellow.)

```
<?xml version="1.0" encoding="UTF-8"?>
<Task schemaVersion="3.0" runNLS="never">
      <Registration>
          <Name>My Combinations Task</Name>
          <Description>This is my first task.</Description>
          <GUID>726B2B12-4542-4200-96F4-960C645D767F</GUID>
            <Procedures>TBD</Procedures>
            <Version>3.3</Version>
            <Links>
            <Link
href="http://support.sas.com/software/products/sasstudio/index.html#s1=2">SAS
Studio Documentation</Link>
            </Links>
      </Registration>
      <Metadata>
            <DataSources>
            </DataSources>


            <Options>
            <Option inputType="string" name="optionsTab">MY OPTIONS</Option>
            <Option name="nObjects" inputType="numstepper" defaultValue="4"
minValue="1" maxValue="100" >Total number of objects:</Option>
            <Option name="nInSet" inputType="numstepper" defaultValue="3"
minValue="1" maxValue="100" >Total number of objects in the set:</Option>
            </Options>
      </Metadata>
        <UI>
            <Container option="optionsTab">
                  <OptionItem option="nObjects"/>
                  <OptionItem option="nInSet"/>
            </Container>
        </UI>
<CodeTemplate>
            <![CDATA[
/* Compute the combinations of n things taken k at a time */
%let n=$nObjects;
%let k=$nInSet;
data combo;
   array c[&k];
   n=&n;
   k=&k;
   if k>n then do;
      put "Error: k must be <= n";
      stop;
   end;
   keep c:;
                                 /* initialize */
   do i=1 to k;
      c[i]=i;
   end;
   output;                       /* output a combination */
                                 /* find next combo */
   do while(1);
      if c[k]<n then c[k]+1;     /* just increment last value */
      else do;
         n1=n;        /* move back down list until one is found */
         found=0;     /* to increment */
         do i=k-1 to 1 by -1 while(found=0);
            n1=n1-1;
            if c[i]<n1 then found=i;
         end;
         if (found=0)then goto term; /* none found ... finished */
         c[found]+1;
         do j=found+1 to k;
            c[j]=c[j-1]+1;
         end;
      end;
      output;                           /* output a combination */
   end;
term:;
run;

            ]]>

      </CodeTemplate>

</Task>
```

ADDING REQUIREMENTS

If you want to make sure your user selects a value for a control, you can add the *required* attribute to the definition of the number stepper control. If the value of the number stepper control is blank, the task will be considered in an invalid state. Most controls honor the *required* attribute. Controls that cannot have an empty state (for example, the slider control) do not support this attribute.

In the Metadata section, find the Option tags. In the tags for both number stepper controls, add required="true". Here is how these tags should look now:

```
<Option name="nObjects" inputType="numstepper" defaultValue="4" minValue="1"
maxValue="100" required="true">Total number of objects:</Option>

<Option name="nInSet" inputType="numstepper" defaultValue="3" minValue="1"
maxValue="100" required="true">Total number of objects:</Option>
```

Run the task. In the task interface, notice the red asterisks that now appear by the number stepper controls. These asterisks indicate that the control is a required control. Now, delete the number in one of the controls. Notice that in the code window the SAS code is gone. Instead, you see a message indicating that your control needs a value and therefore your task is invalid.

Sometime requirements can get more complicated. The CTM can have a Requirements section for these complex situations as well. Between the UI section and the Code Template section, add the following code:

```
<Requirements>

            <Requirement condition="$nInSet &lt;= $nObjects">

                  <Message>The number of objects in a set must be less than
or equal to the total number of objects.</Message>

            </Requirement>

      </Requirements>
```

Notice the condition statement is contained within the Requirement tag. This condition statement uses the same Velocity variables that are used in the Code Template section. In XML, some symbols (such as <,>, and &) have special meaning. Because these symbols are common in conditional expressions, it is important to remember to use the XML entity references (&lt; &gt; &amp;) instead of the symbols.

If the condition resolves to false, the task is considered in an invalid state, and the message that you specified appears in the code window.

Run your task. Now, change the total number of objects to be less than the number of objects in a set. Instead of code in the code window, you now see the message you specified in the previous Requirements section.

ADDING DEPENDENCIES

The SAS code at the beginning of this paper stores the results in an output data set named COMBO. You will now add a control that enables the user to specify the name for the output data set. To demonstrate how a dependency works, you will also add a check box that enables or disables the output data set control.

Recall that dependencies allow for the state of one control to affect the state of another controls. Controls can be enabled, disabled, and made visible or invisible. Many controls can have their value changed as the result of a dependency.

For this example, your dependency simply enables or disables a control based on the value of a check box.

Add the following two controls to the Metadata section.

```
<Option name="createOutput" inputType="checkbox" defaultValue="0">Create
output data set</Option>

<Option name="outputDS" indent="1" inputType="outputdata"
defaultValue="Combo" required="true">Data set name: </Option>
```

Now add this XML to the UI section.

```
<OptionItem option="createOutput"/>

<OptionItem option="outputDS"/>
```

You are getting the hang of this!

Now, you are ready to add the dependency. Immediately after your Requirements section, add the following code:

```
<Dependencies>

   <Dependency condition="($createOutput == '1')">

      <Target conditionResult="true" option="outputDS" action="enable"/>

      <Target conditionResult="false" option="outputDS" action="disable"/>

   </Dependency>

</Dependencies>
```

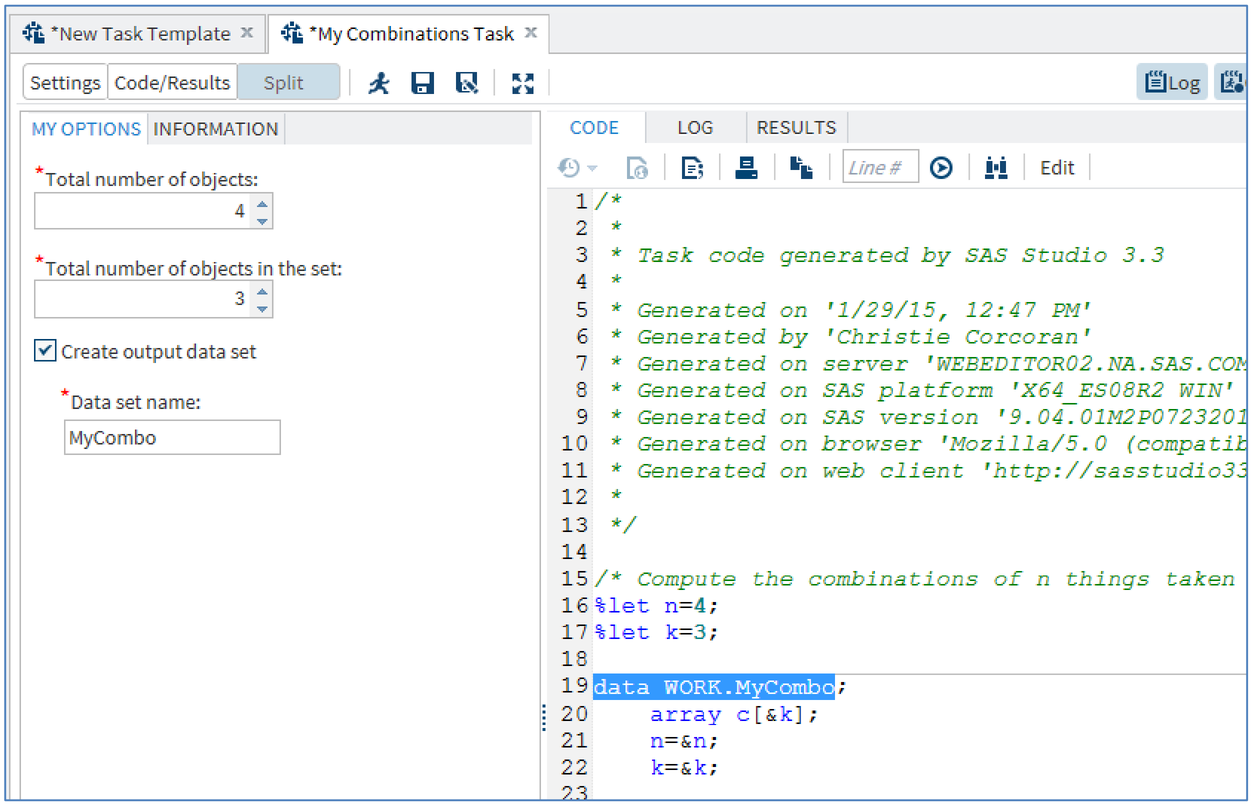
Run your task. Notice that the output data set control enables or disables based on whether the check box is selected. Also notice that your SAS code is unaffected when you modify the contents of the output data set control. The Velocity script in the Code Template section needs to be modified to use these new controls.

Make the highlighted changes to the Code Template section:

```
/* Compute the combinations of n things taken k at a time */

%let n=$nObjects;

%let k=$nInSet;

#if ($createOutput == 1) #set($outputName = $outputDS)

#else #set($outputName = "combo") #end

data $outputName;

   array c[&k];
   n=&n;
   k=&k;
   if k>n then do;
      put "Error: k must be <= n";
      stop;
   end;
   keep c:;
                              /* initialize */
   do i=1 to k;
      c[i]=i;
   end;
   output;                    /* output a combination */
                              /* find next combo */
   do while(1);
      if c[k]<n then c[k]+1;  /* just increment last value */
      else do;
         n1=n;        /* move back down list until one is found */
         found=0;     /* to increment */
         do i=k-1 to 1 by -1 while(found=0);
            n1=n1-1;
            if c[i]<n1 then found=i;
         end;
         if (found=0)then goto term; /* none found ... finished */
         c[found]+1;
         do j=found+1 to k;
            c[j]=c[j-1]+1;
         end;
      end;
      output;                          /* output a combination */
   end;
term:;
run;
```

Run your task again. This time, enable the output data set control and change the name of the output data set. This name can be in the form Library.Member, or if just a Member name is specified, the Work library will be used.

The screenshot in Figure 6 shows what your final task should look like. In the SAS code, the DATA step is highlighted to show the changes that you made using the output data set control.



*Total number of objects: 4

*Total number of objects in the set: 3

☑ Create output data set

*Data set name: MyCombo

```
/*
 *
 * Task code generated by SAS Studio 3.3
 *
 * Generated on '1/29/15, 12:47 PM'
 * Generated by 'Christie Corcoran'
 * Generated on server 'WEBEDITOR02.NA.SAS.COM
 * Generated on SAS platform 'X64_ES08R2 WIN'
 * Generated on SAS version '9.04.01M2P0723201
 * Generated on browser 'Mozilla/5.0 (compatib
 * Generated on web client 'http://sasstudio33
 *
 */

/* Compute the combinations of n things taken
%let n=4;
%let k=3;

data WORK.MyCombo;
    array c[&k];
    n=&n;
    k=&k;
```

Figure 6: Screenshot of Completed Task

YOU DID IT – WHAT'S NEXT?

Congratulations! You are well on your way to writing your own tasks. Although this was a simple task, it demonstrated each of the sections in a CTM. Now that you know the basics, where do you go from here?

There are a number of places you can go to learn more about writing CTMs. The *SAS Studio: Developer's Guide to Writing Custom Tasks* is a good resource for learning about all of the input types and their associated attributes. This document can be accessed at http://support.sas.com/documentation/onlinedoc/sasstudio/index.html. It also goes into greater detail on other sections, such as Requirements and Dependencies.

To create your task, you selected the **New Task** option from the drop-down menu. Notice that in this menu there is also a **Sample Task Template**. This option opens a task that provides very basic samples of most CTM controls. Also note the **Advanced Task Template** option. This option opens a task demonstrating some of the more complex controls and CTM functionality. Both the Sample Task Template and the Advanced Task Template are good resources for learning about the CTM.

Another trick for learning how to write your own tasks is to look at the CTM of some of our built-in tasks. Although you cannot modify the CTM for the built-in tasks directly, you can copy these tasks to My Tasks or a Folder. You can then edit them directly through SAS Studio or by using another editing applications, such as Notepad++. To copy a task, right-click the task that you are interested in. From the pop-up menu, select either **Add to My Tasks** or **Add to Folders**.

Although a bit different from the task you just created, Dr. Goodnight's task is available in SAS Studio. You'll find his Combinations task in the Combinatorics and Probability category. In fact, in SAS Studio 3.3, all of the tasks in that category were designed by him.

SHARING YOUR TASKS

You have a created a task that runs the report everyone keeps asking you to run for them. You have even added dependencies and requirements, so they cannot generate any bad SAS code. Now, you are ready to share the task. You already know that you can save a task as a CTM file, so sharing is as easy as making that file available to others. You can e-mail the CTM file or put it in a shared file location that everyone can access. They just access the CTM file from their own SAS Studio session and run your task. Of course, they will need all the normal permissions – access to SAS Studio, access to your file, and the authority to run whatever SAS code the task generates. It's not that different from sharing a normal SAS program, but now you have a much easier-to-use web-based interface that everyone can use. You can stop teaching everyone about SAS macro variables!

In some cases, it's nice to give your users a starting point – a task that will run as-is without them having to select anything. For example, when you develop a task, you might want to include a default input data source or default option settings for the users at your site. In SAS Studio, you can save a task as a CTK file. When users at your site run this CTK file, they see your default settings.

CONCLUSION

SAS Studio is shipped with several predefined tasks, which are point-and-click user interfaces that guide the user through an analytical process. When a user selects a task option, SAS code is generated and runs on the SAS server. Any output (such as graphical results or data) is displayed in SAS Studio.

Because of the flexibility of the task framework, you can create tasks for your site. In SAS Studio, all tasks use the same Common Task Model (CTM) and the Velocity Template Language. The CTM defines the template for the task - how the task appears to the SAS Studio user and the code that is needed to run the task. The elements of the CTM enable you to control the user interface, set up dependencies and requirements, and provide messages and documentation. In essence, all the tools you need to create a basic web-based front end to your SAS code for yourself or to share with others.

CONTACT INFORMATION

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

Christie Corcoran
100 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
Christie.Corcoran@sas.com
http://www.sas.com

Amy Peters
100 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
Amy.Peters@sas.com
http://www.sas.com