Paper SAS4381-2016

Create Web-Based SAS® Reports Without Having to Be a Web Developer

Marie Dexter, Kris Kiser, Amy Peters, and Christie Corcoran, SAS Institute Inc., Cary, NC

ABSTRACT

You might already know that SAS® Studio tasks provide you with prompts to “fill in the blanks” in SAS® code and help you navigate SAS syntax. But did you know that SAS Studio tasks are not only designed to allow you to modify them, but that there’s an entire common task model (CTM) provided for you to build your own? You can build basic utilities or complex reports. You can just run SAS code or you can have elaborate prompts to dynamically generate code. And since SAS Studio runs in a web browser, your tasks are browser-based and can be easily shared with others. In this paper you will learn how to take a typical SAS program and create a SAS Studio task from it. When the task is run, you’ll see web-based prompts for the dynamic portions of the program and HTML output delivered to your browser.

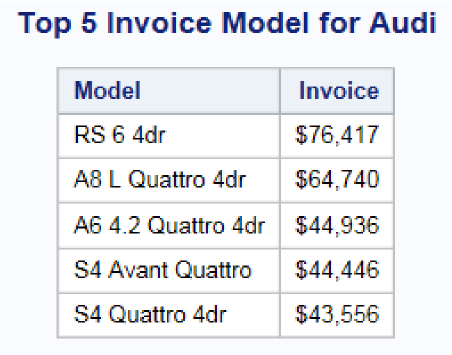
INTRODUCTION

As a SAS programmer, you are often asked to run reports that rank items according to sales, invoice values, and so on. Using the Sashelp.Cars data set as the input data source, you recently wrote the following SAS program to rank the top 5 models of Audi cars by their invoice value.

To generate the report in SAS Studio, copy and paste the following code into the code editor:

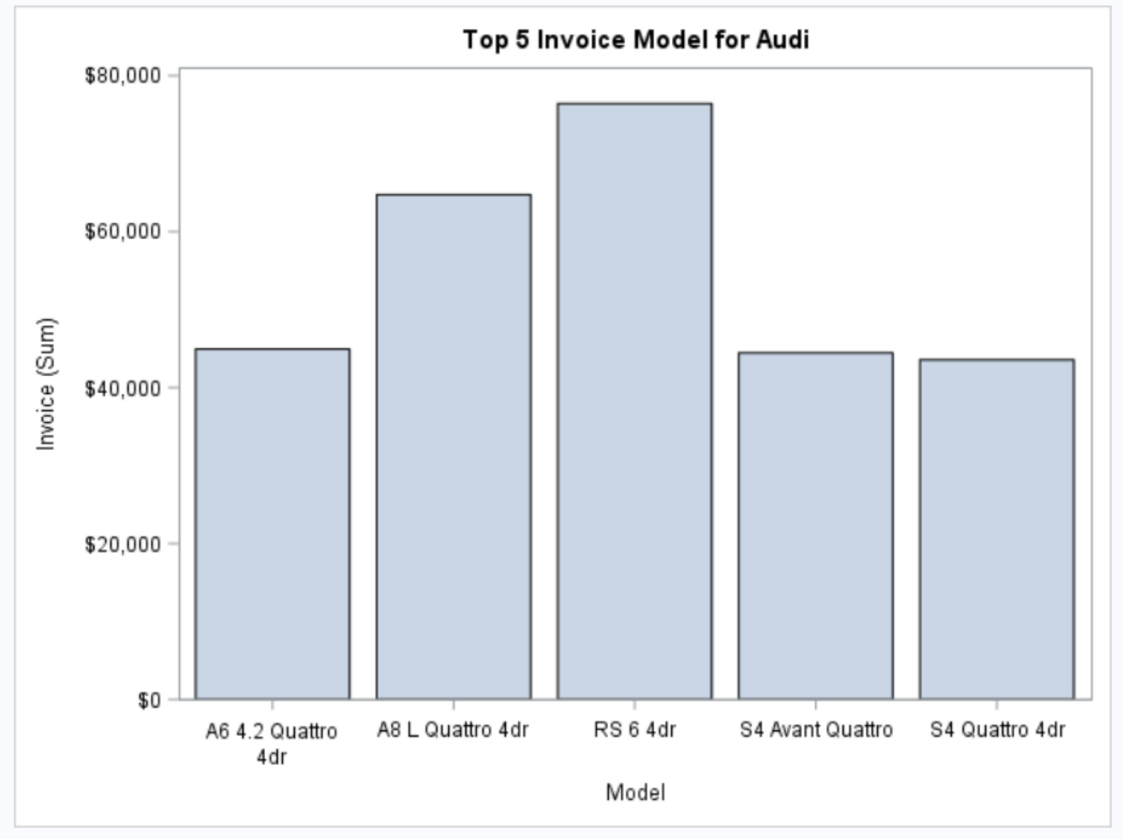
```
%let rankNumber=5;
proc rank data=sashelp.cars(where=(make="Audi"))descending ties=mean
out=work.rank(where=(rank<=&rankNumber));
   var invoice;
   ranks rank;
run;
proc sql ;
      create table work.toprank as
         select model, invoice
            from work.rank
            where rank<=&rankNumber
            order by rank;
quit;
title "Top &rankNumber Invoice Model for Audi";
proc print data=work.toprank noobs label;
run;
proc sgplot data=work.toprank;
      vbar model / response=invoice;
run;
```

Click 🏃 to run the SAS program. The results appear on the **RESULTS** tab in SAS Studio.



Top 5 Invoice Model for Audi

Model	Invoice
RS 6 4dr	$76,417
A8 L Quattro 4dr	$64,740
A6 4.2 Quattro 4dr	$44,936
S4 Avant Quattro	$44,446
S4 Quattro 4dr	$43,556



Display 1. Tabular and Graphical Results for the Top 5 Models of Audi Ranked by Invoice Value

Other users at your site are asking you to customize this report to meet their needs. For example, one user wants a report that contains 10 ranked values. Another user wants to rank the data based on the car's make rather than the model.

By converting your SAS program into a SAS Studio task, other users (who might not be SAS programmers) can use a point-and-click interface to create the report they want without asking for your help.

WHAT IS A SAS STUDIO TASK?

SAS Studio is shipped with several predefined tasks, which are point-and-click user interfaces that guide a user through an analytical process. 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. No Java programming or ActionScript programming is required to build a task.

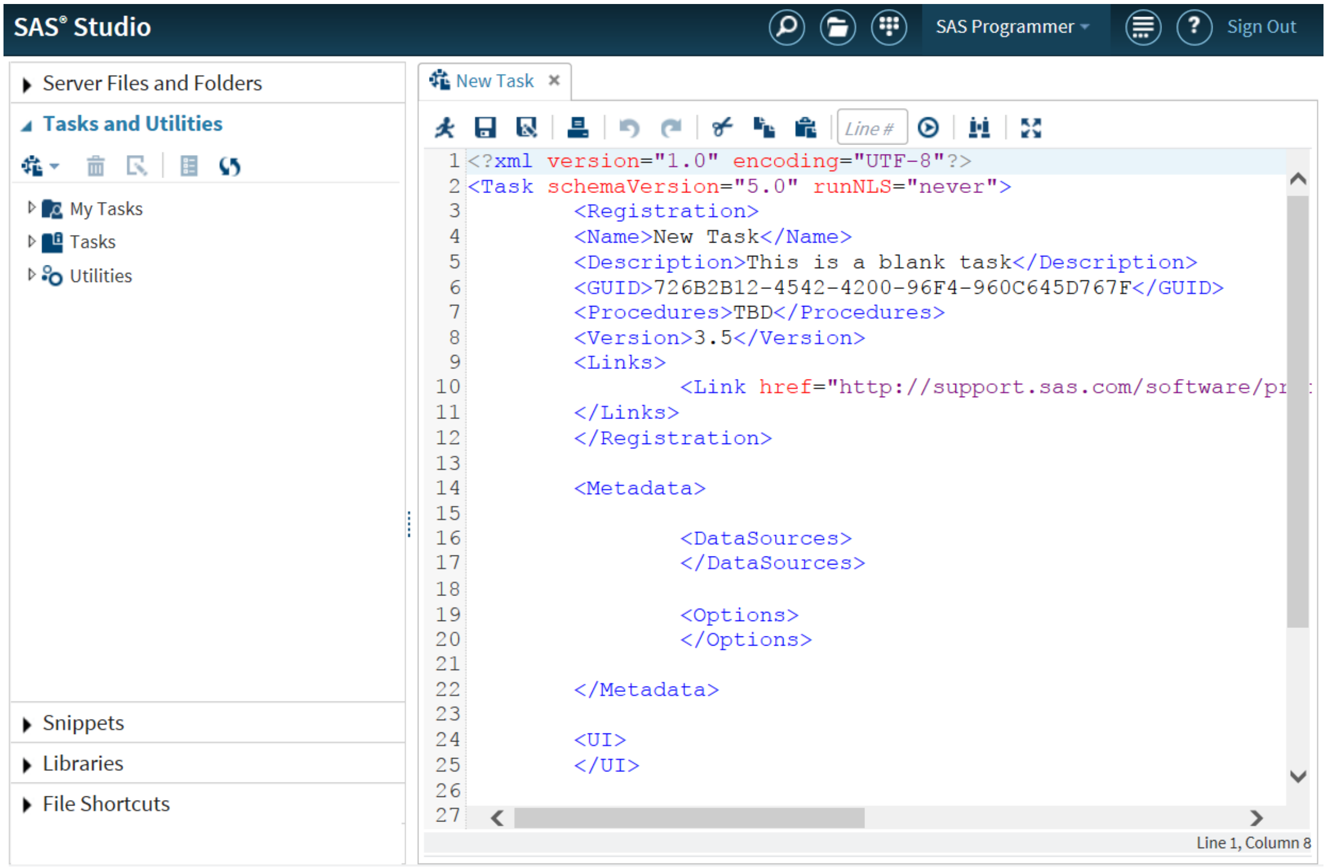
The common task model defines the XML template for the task. In the CTM file, you define how the task appears to the SAS Studio user and specify the code that is needed to run the task. A task is defined by its input data and the options that are available to the user. (Some tasks might not require an input data source.) In addition, the task has metadata so that it is recognized by SAS Studio.

CREATE YOUR CUSTOM TASK

STEP 1: OPEN A BLANK TASK

Open SAS Studio. In the navigation pane, click **Tasks and Utilities**. To open a blank task, click and select **New Task**.

A blank task opens in the SAS Studio workspace.



```
<?xml version="1.0" encoding="UTF-8"?>
<Task schemaVersion="5.0" runNLS="never">
        <Registration>
        <Name>New Task</Name>
        <Description>This is a blank task</Description>
        <GUID>726B2B12-4542-4200-96F4-960C645D767F</GUID>
        <Procedures>TBD</Procedures>
        <Version>3.5</Version>
        <Links>
                <Link href="http://support.sas.com/software/pr
        </Links>
        </Registration>

        <Metadata>

                <DataSources>
                </DataSources>

                <Options>
                </Options>

        </Metadata>

        <UI>
        </UI>

```

Display 2. Template for a Blank Task

STEP 2: COPY YOUR CODE INTO THE CODE TEMPLATE

Scroll to the end of the blank task and locate the `CodeTemplate` element. This element contains the SAS code required to generate the HTML results.

Here is the default code in the `CodeTemplate` element:

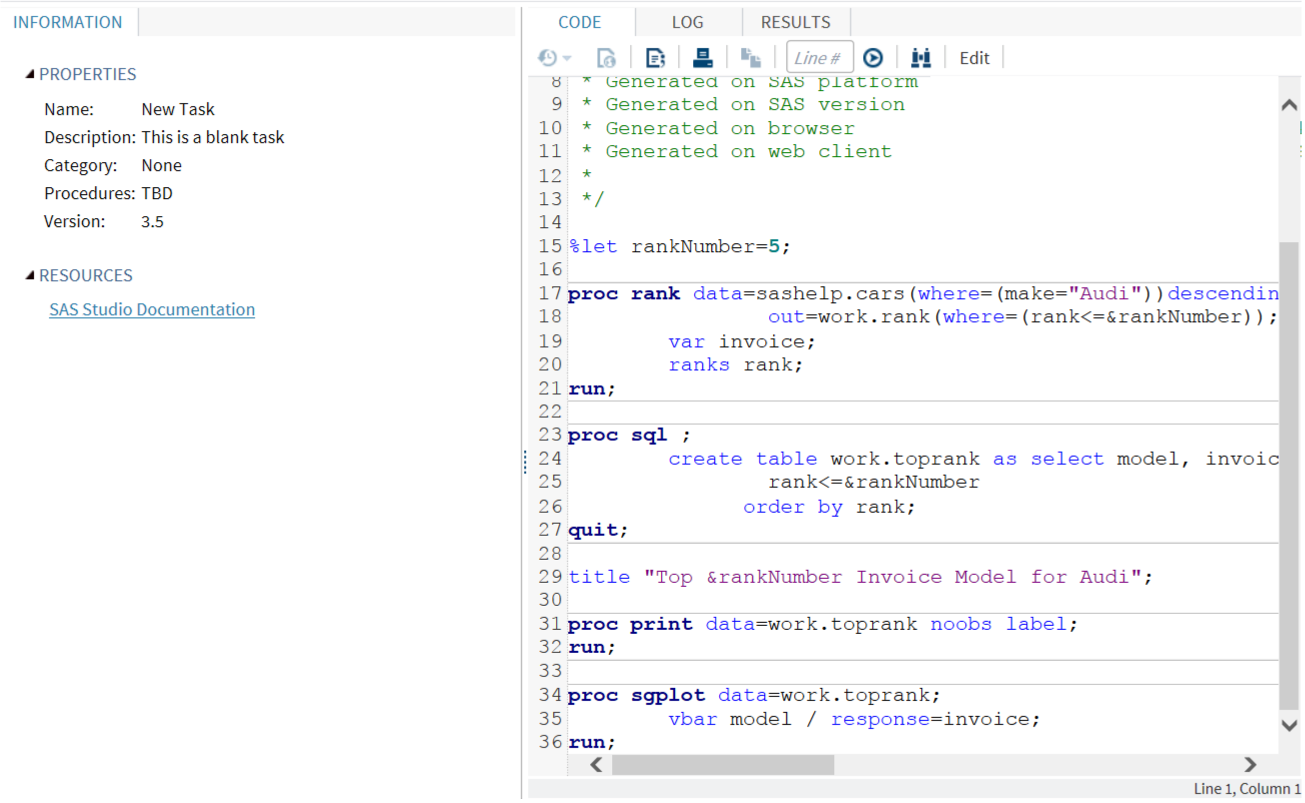
```
<CodeTemplate>
   <![CDATA[
      proc print data=sashelp.cars;run;
   ]]>
</CodeTemplate>
```

If you ran this task now, the HTML results would show the contents of the Sashelp.Cars data set.

Replace the PROC PRINT statement with your SAS program. Here is the `CodeTemplate` element with your SAS program:

```
<CodeTemplate>
            <![CDATA[

%let rankNumber=5;
proc rank data=sashelp.cars(where=(make="Audi"))descending ties=mean
   out=work.rank(where=(rank<=&rankNumber));
   var invoice;
   ranks rank;
run;
proc sql ;
      create table work.toprank as
         select model, invoice
            from work.rank
            where rank<=&rankNumber
            order by rank;
quit;
title "Top &rankNumber Invoice Model for Audi";
proc print data=work.toprank noobs label;
run;
proc sgplot data=work.toprank;
      vbar model / response=invoice;
run;

            ]]>
</CodeTemplate>
```

Click 🏃 to generate the task interface. A **New Task** tab appears in the workspace. On the left, you see an **INFORMATION** tab. (You can ignore this for now.) On the right, the **CODE** tab shows the SAS code that you entered into the `CodeTemplate` element.



PROPERTIES

Name: New Task
Description: This is a blank task
Category: None
Procedures: TBD
Version: 3.5

RESOURCES

SAS Studio Documentation

```
 * Generated on SAS platform
 * Generated on SAS version
 * Generated on browser
 * Generated on web client
 *
 */

%let rankNumber=5;

proc rank data=sashelp.cars(where=(make="Audi"))descendin
                out=work.rank(where=(rank<=&rankNumber));
        var invoice;
        ranks rank;
run;

proc sql ;
        create table work.toprank as select model, invoic
                rank<=&rankNumber
              order by rank;
quit;

title "Top &rankNumber Invoice Model for Audi";

proc print data=work.toprank noobs label;
run;

proc sgplot data=work.toprank;
        vbar model / response=invoice;
run;
```

Display 3. Your SAS Program as the Basis for a SAS Studio Task

Click again to run the task. The HTML results appear on the **RESULTS** tab. Because you are using the same SAS code, the results are identical to when you ran the SAS program in the code editor. Close the **New Task 1** tab.

To save your new SAS Studio task, open the tab that contains your task code and click . In the Save As window, select **My Tasks**. For the task name, enter `Rank_Cars`. Verify that the **Save as type** is **SAS Studio Task (*.CTM)** and click **Save**.

STEP 3: REPLACE THE RANK MACRO VARIABLE WITH AN OPTION IN THE TASK INTERFACE

In your SAS program, you use a macro variable, `rankNumber`, to specify the number of values to rank in the HTML results. Currently to change the value of `rankNumber`, you must edit the SAS code.

However, everyone at your site isn't familiar with SAS programming. Wouldn't it be nice if you could create a point-and-click option instead? In this step, you will add the **Rank Number** option to your task.

Before you can include them in the task interface, all tabs, headings, and options must first be defined in the `Metadata` element. In this step, you want to create a **Top Rank** tab (which acts as the container), an **OPTIONS** group on this tab, and an option called **Rank Number** in this group.

In the Rank_Cars task, add the highlighted code to the `Options` element:

```
<Metadata>

   <DataSources>
   </DataSources>

   <Options>
      <Option name="topRankTab" inputType="string">Top Rank</Option>
      <Option name="optionsGroup" inputType="string">OPTIONS</Option>
      <Option name="topRankNumber" inputType="numstepper" minValue="0"
         maxValue="10" required="true" defaultValue="5">Rank Number</Option>
   </Options>

</Metadata>
```

Let us examine this code:

1. The first `Option` element defines the **Top Rank** tab. The name, `topRankTab`, is used to reference this tab throughout the task code.
2. The second `Option` element defines the **OPTIONS** group on the **Top Rank** tab. The name, `optionsGroup`, is used to reference this heading throughout the task code.
3. The third `Option` element defines the **Rank Number** option.
   - The name, `topRankNumber`, is used to reference this option throughout the task code.
   - The `inputType` parameter specifies the type of control for the user interface. This example uses a numstepper control. (For more information about the types of controls, see *SAS Studio: Developer’s Guide for Writing Custom Tasks* at http://support.sas.com/documentation/onlinedoc/sasstudio.)
   - The `defaultValue` parameter specifies the option’s initial value. In this case, the default value is `5`.
   - The `minValue` and `maxValue` parameters specify the range of valid values. For this example, the minimum value is `0` and the maximum value is `10`.
   - Because the `required` parameter is set to `true`, you must enter a value for the **Rank Number** option to run the task.

After defining the **Top Rank** tab, the **OPTIONS** group, and the **Rank Number** option in the metadata, you must add these items to the `UI` element so that they will appear in the task interface.

In the Rank_Cars task, add the highlighted code to the `UI` element:

```
<UI>
   <Container option="topRankTab" open="true">
      <Group option="optionsGroup" open="true">
         <OptionItem option="topRankNumber"/>
      </Group>
   </Container>
</UI>
```

Here is an explanation of this code:

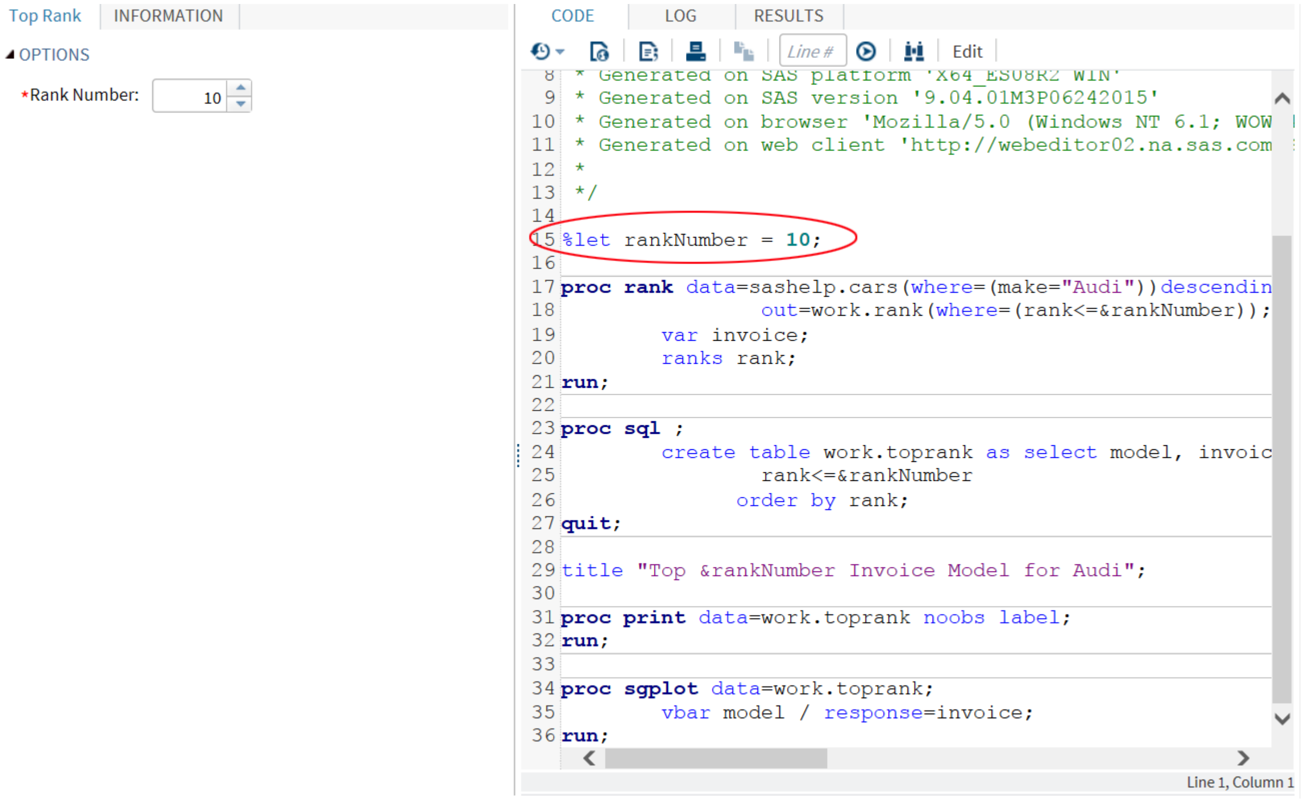
1. The `Container` element creates the user interface for the **Top Rank** tab. Note that the `option` parameter is set to `topRanktab`. Because the `open` parameter is set to `true`, the **Top Rank** tab is visible when you open the task interface.
2. The `Group` element creates the **OPTIONS** group heading on the **Top Rank** tab. The `option` parameter is set to `optionsGroup`. Because the `open` parameter is set to `true`, this group is open when you open the task interface. You can expand and collapse group headings.
3. The `OptionItem` element specifies the placement of the **Rank Number** option. In the task interface, this option appears on the **Top Rank** tab (the container) in the **OPTIONS** group. The option parameter is set to `topRankNumber`.

The last step is to update the macro code in the `CodeTemplate` element to refer to the new `topRankNumber` option. In the %LET statement, replace `5` with `$topRankNumber`. This Velocity variable contains the value that the user selected for the `topRankNumber` option (which is the **Rank Number** option in the task interface).

```
<CodeTemplate>
   %let rankNumber = $topRankNumber;
   proc rank data=sashelp.cars(where=(make="Audi"))descending ties=mean
      out=work.rank(where=(rank<=&rankNumber));
      var invoice;
  …
   run;
<CodeTemplate>
```

Click 🏃 to generate the task interface. You should see a **Top Rank** tab that contains the **OPTIONS** group heading and the **Rank Number** option.

On the **Top Rank** tab, change the **Rank Number** option to `10`. On the **CODE** tab, you see that `%let rankNumber` now equals `10`.

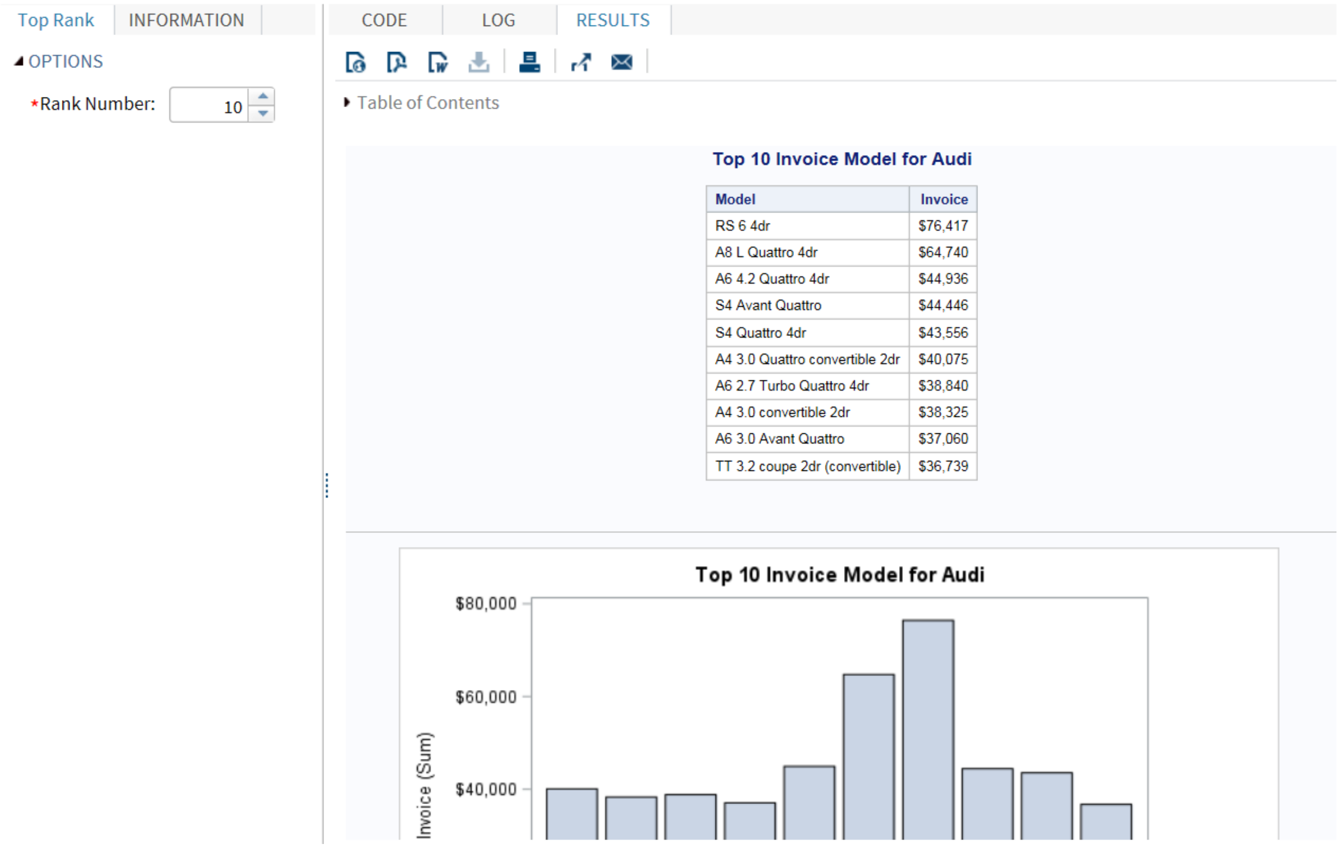


*Rank Number: 10

```
* Generated on SAS platform 'X64_ES08R2 WIN'
* Generated on SAS version '9.04.01M3P06242015'
* Generated on browser 'Mozilla/5.0 (Windows NT 6.1; WOW
* Generated on web client 'http://webeditor02.na.sas.com
*
*/

%let rankNumber = 10;

proc rank data=sashelp.cars(where=(make="Audi"))descendin
                out=work.rank(where=(rank<=&rankNumber));
        var invoice;
        ranks rank;
run;

proc sql ;
        create table work.toprank as select model, invoic
                rank<=&rankNumber
              order by rank;
quit;

title "Top &rankNumber Invoice Model for Audi";

proc print data=work.toprank noobs label;
run;

proc sgplot data=work.toprank;
        vbar model / response=invoice;
run;
```

Display 4. Change the Value of the Rank Number Option

Click again to run the task. The **RESULTS** tab shows an HTML report that contains 10 ranked values.



Top 10 Invoice Model for Audi

Model	Invoice
RS 6 4dr	$76,417
A8 L Quattro 4dr	$64,740
A6 4.2 Quattro 4dr	$44,936
S4 Avant Quattro	$44,446
S4 Quattro 4dr	$43,556
A4 3.0 Quattro convertible 2dr	$40,075
A6 2.7 Turbo Quattro 4dr	$38,840
A4 3.0 convertible 2dr	$38,325
A6 3.0 Avant Quattro	$37,060
TT 3.2 coupe 2dr (convertible)	$36,739

Display 5. Top 10 Invoice Report in SAS Studio

STEP 4: ADDING A SUBSET VARIABLE AND A FILTER

Currently, the report ranks the invoice values for the Audi model. However, you'd like to enable the task user to select a different make (such as BMW) or type (such as hybrid or sedan) of vehicle. You can do this by adding a subset variable to the task.

To allow users to select a subset variable, you must define a role for the subset variable in the metadata. In the Rank_Cars task, locate the `DataSources` element and add the highlighted code:

```
<DataSources>
   <DataSource name="dataset">
      <Roles>
         <Role name="subsetRole" type="C" minVars="1" maxVars="1"
            required="true">Subset by:</Role>
      </Roles>
   </DataSource>
</DataSources>
```

Here is an explanation of this code:

- The `DataSource` element defines the name of the input data source. This name is referenced in the `CodeTemplate` element.

In this paper, the input data source is always Sashelp.Cars. Changing the input data source is beyond the scope of this paper. For more information about data sources, see *SAS Studio 3.5: Developer's Guide for Writing Custom Tasks*.

- The `Role` element enables you to select a variable from the input data source. The ranked report is created using the character variable (`type=C`) assigned to this role. The `minVars` parameter is set to `1`, so you must assign one variable to this role. The `maxVars` parameter is set to `1`, so you cannot assign more than one variable.

Next, you need to define an option, so the task user can select the specific make or type of vehicle.

In the `Options` element, add the highlighted code:

```
<Options>
   <Option name="topRankTab" inputType="string">Top Rank</Option>
   <Option name="dataGroup" inputType="string">DATA</Option>
   <Option name="optionsGroup" inputType="string">OPTIONS</Option>
   <Option name="topRankNumber" inputType="numstepper" minValue="0"
      maxValue="10" required="true" defaultValue="5">Rank Number:</Option>
   <Option name="filterValue" inputType="distinct" required="true"
      source="subsetRole">Where value equals:</Option>
</Options>
```

Here is an explanation of this code:

1. In the second `Option` element, you are defining a **DATA** group.
2. In the fifth `Option` element, you are defining the **Where value equals** option. You will use this option to select the value of interest for the variable assigned to the **Subset by** role. Because `inputType=distinct`, the **Where value equals** option is a drop-down list that contains only unique values. (No duplicate values appear in the drop-down list.) Because `source=subsetRole`, the `subsetRole` option (or **Subset by** role in the task interface) contains the values for this drop-down list.

Now, you must add the new **DATA** group, the **Subset by** role, and the **Where value equals** option to the `UI` element:

```
<UI>
   <Container option="topRankTab" open="true">
      <Group option="dataGroup" open="true">
         <DataItem data="dataset"/>
         <RoleItem role="subsetRole"/>
         <OptionItem option="filterValue"/>
      </Group>
      <Group option="optionsGroup" open="true">
         <OptionItem option="topRankNumber"/>
      </Group>
   </Container>
</UI>
```

Here is an explanation of this code:

1. The `Group` element creates the new **DATA** group heading on the **Top Rank** tab.
2. The `DataItem` element displays the name of the input data source. This paper uses Sashelp.Cars as the input data set.
3. The `RoleItem` element displays the **Subset by** role.
4. The `OptionItem` element displays the **Where value equals** option.

Only one more step! Now, you must update the code in the `CodeTemplate` element. The names that you defined in the `Role` and `Option` elements are used in the Velocity code.

Make the highlighted changes to the `CodeTemplate` element:

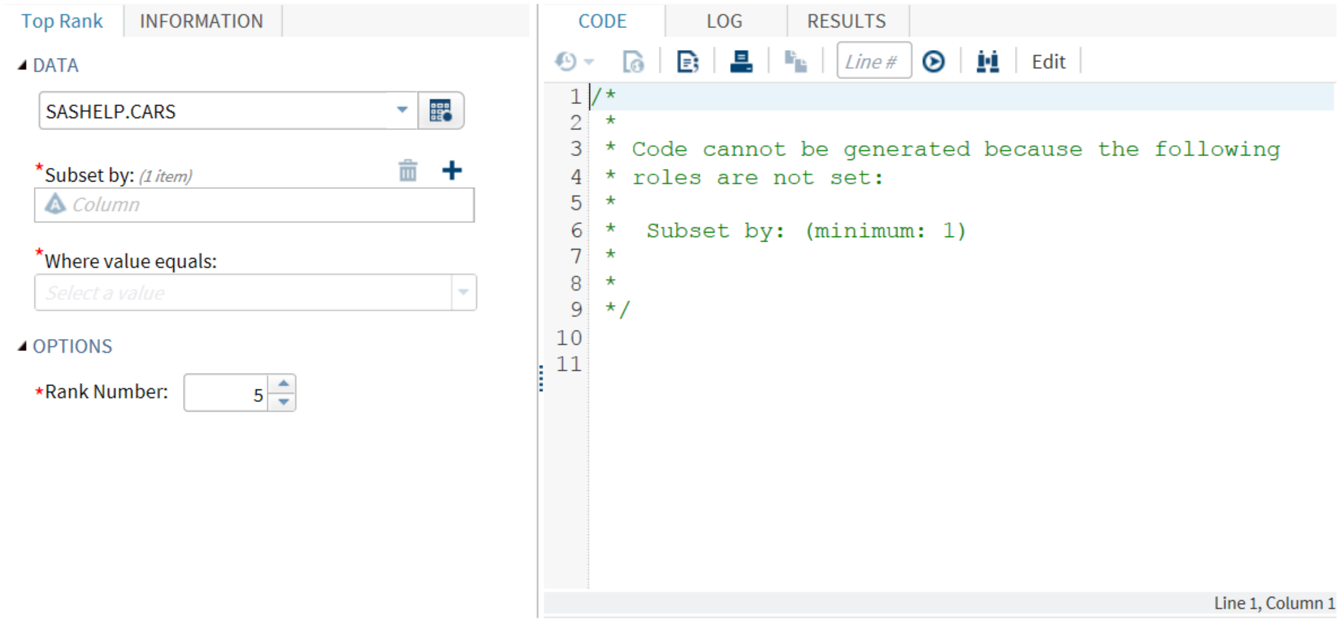
```
<CodeTemplate>
   %let rankNumber = $topRankNumber;
   proc rank data=$dataset (where=($subsetRole[0]="$filterValue"))descending
      ties=mean
      out=work.rank(where=(rank<=&rankNumber));
      var invoice;
      ranks rank;
   run;
   proc sql ;
      create table work.toprank as
      select model, invoice
      from work.rank
      where rank<=&rankNumber
      Order by rank;
   quit;
   title "Top &rankNumber Invoice Model for $filterValue";
   proc print data=work.toprank noobs label;
   run;
   proc sgplot data=work.toprank;
      vbar model / response=invoice;
   run;
</CodeTemplate>
```

Here are the changes to the PROC RANK statement:

1. The DATA option needs to refer to the `$dataset` Velocity variable. The value of the `$dataset` Velocity variable depends on what data source is selected in the task interface. Remember for this paper select Sashelp.Cars as the data source.
2. In the original SAS program, `where=(make=”Audi”)`. With the addition of the **Subset by** role and the **Where value equals** option, the task user can now change the values for the report, so you need to replace the code in parenthesis with Velocity variables.
    - The `$subsetRole` Velocity variable is an array that contains the variable selected for the **Subset by** role. The `Role` element uses the `minVars` and `maxVars` parameters to specify how many variables can be assigned to a role. Because roles can have 1 to *n* number of variables, the corresponding Velocity variable is an array and must be followed by `[0]` in the `CodeTemplate` element.
    - The `$filterValue` Velocity variable contains the value selected from the **Where value equals** drop-down list.

You can also use the `$filterValue` Velocity variable in the TITLE statement, so the value you are filtering by appears in the title of the results.

Click 🏃 to generate the task interface. On the **Top Rank** tab, you see the new **DATA** group, a data set selector (which should be set to Sashelp.Cars), the **Subset by** role, and the **Where value equals** option.



SASHELP.CARS

*Subset by: *(1 item)*

Column

*Where value equals:

Select a value

*Rank Number: 5

```
/*
 *
 * Code cannot be generated because the following
 * roles are not set:
 *
 *  Subset by: (minimum: 1)
 *
 *
 */
```

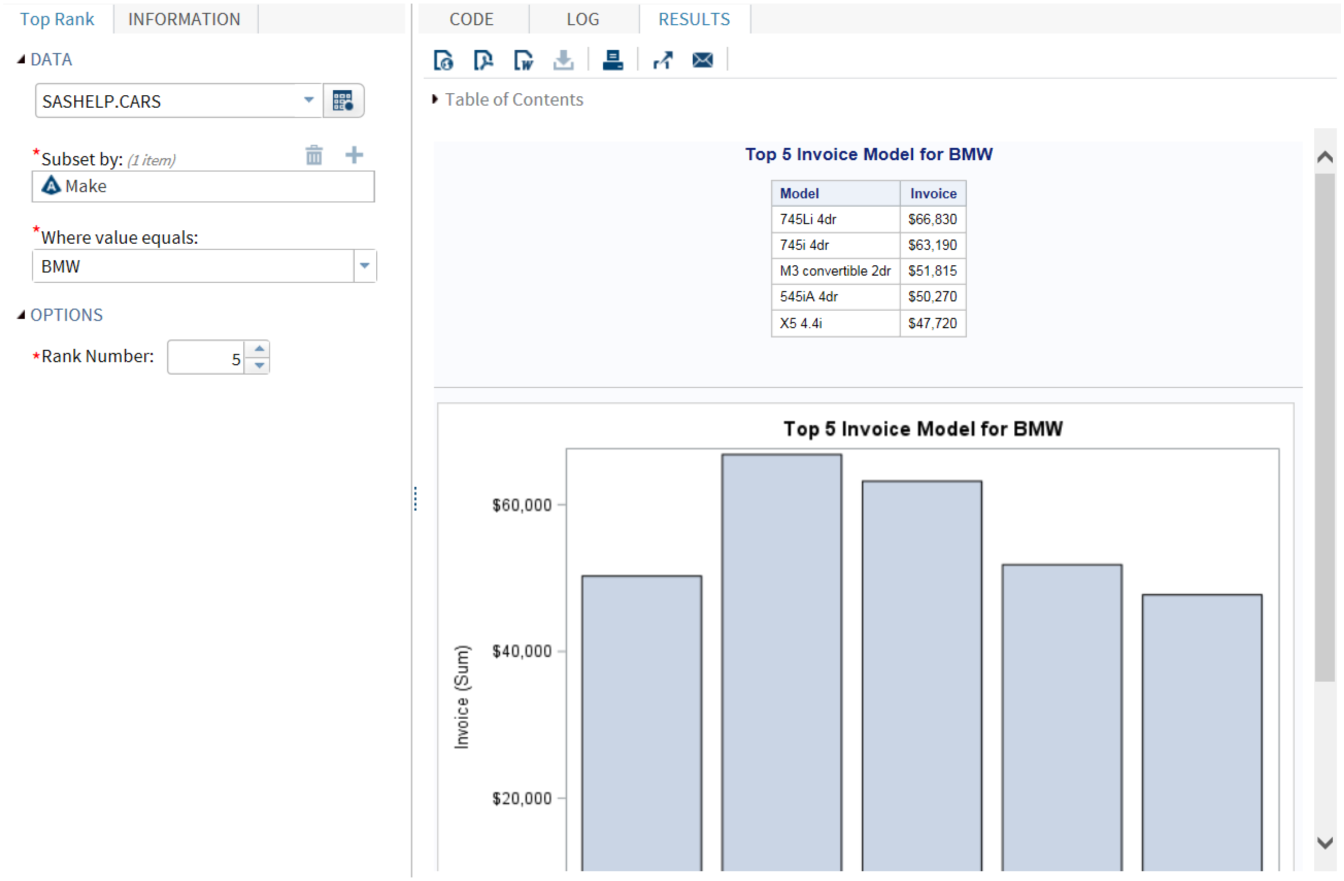
Line 1, Column 1

Display 6. Subset by Role and Where value equals Option in the Task Interface

Initially, a message (rather than code) appears on the **CODE** tab. The red asterisks in the task interface indicate that you must assign a variable to the **Subset by** role and select a value from the **Where value equals** drop-down list before you can run the task.

For the **Subset by** role, click ✚ and select **Make**. The values in the **Where value equals** drop-down list now contain all the distinct values of Make. From the **Where value equals** drop-down list, select **BMW**. Leave 5 as the default value for the **Rank Number** option.

Click 🏃 to run the task. The results show a report of the top 5 makes of BMW based on their invoice value.



DATA

SASHELP.CARS

*Subset by: *(1 item)*

Make

*Where value equals:

BMW

OPTIONS

*Rank Number: 5

Top 5 Invoice Model for BMW

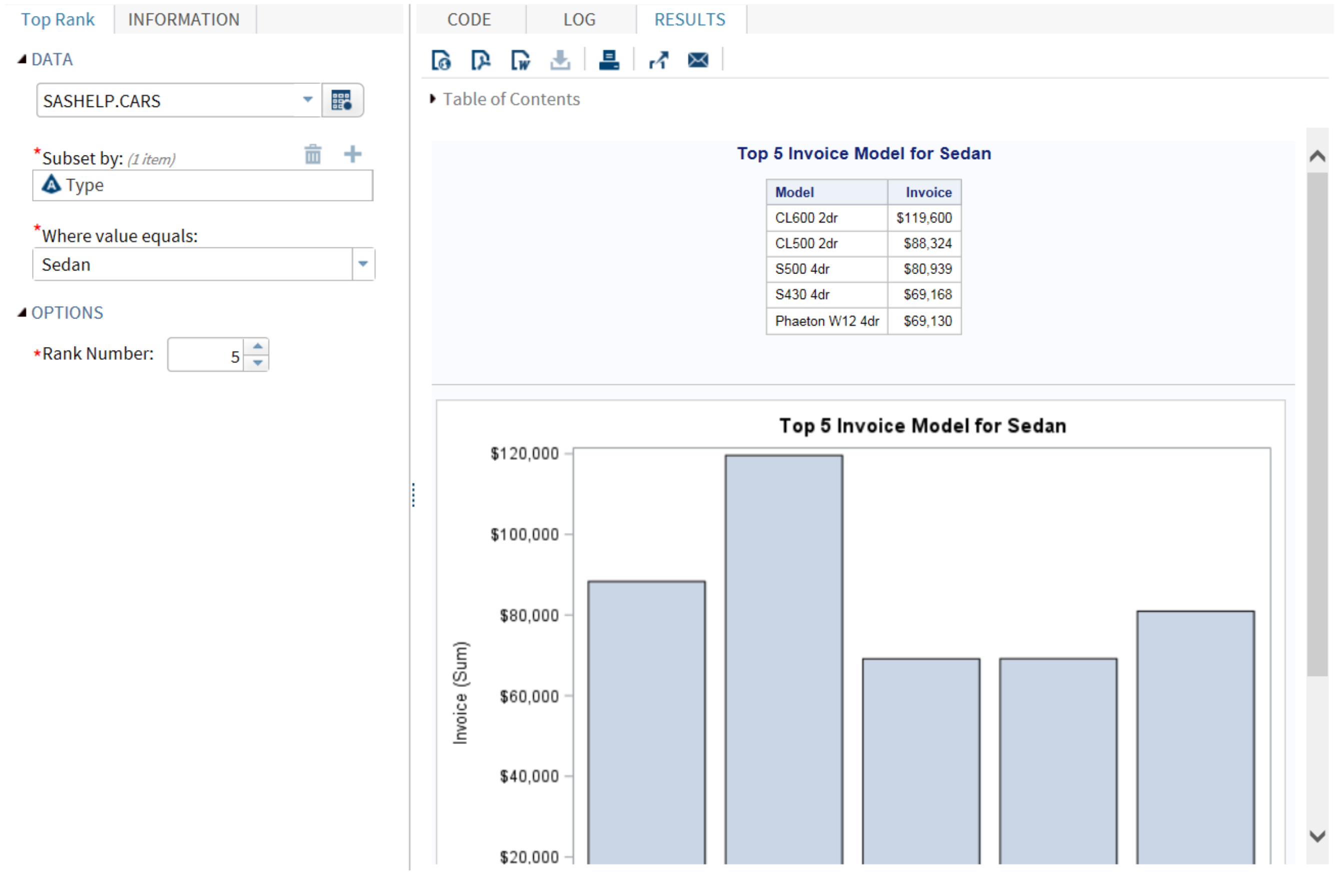
Model	Invoice
745Li 4dr	$66,830
745i 4dr	$63,190
M3 convertible 2dr	$51,815
545iA 4dr	$50,270
X5 4.4i	$47,720

Top 5 Invoice Model for BMW

Display 6. Top 5 Invoice Models for BMW

Let us change these options to generate a report that subsets by type. On the **Top Rank** tab, select the **Subset by** role and click 🗑 . The Make variable is deleted from this role. Next, click ✚ and select **Type**. The values in the **Where value equals** drop-down list now contain the values for the Type variable. From this drop-down list, select **Sedan**.

Click 🏃 to run the task. The results show a report of the top 5 sedans ranked by their invoice value.



DATA

SASHELP.CARS

*Subset by: *(1 item)*

Type

*Where value equals:

Sedan

OPTIONS

*Rank Number: 5

Top 5 Invoice Model for Sedan

Model	Invoice
CL600 2dr	$119,600
CL500 2dr	$88,324
S500 4dr	$80,939
S430 4dr	$69,168
Phaeton W12 4dr	$69,130

Display 7. Top 5 Invoice Models for Sedans

Congratulations! You’ve successfully converted your SAS program into a SAS Studio task.

CONCLUSION

SAS Studio tasks provide a point-and-click interface to help you run analyses. Using the tools of the common task model (CTM) in SAS Studio, you can create custom tasks based on the needs of your site. You can use your existing SAS programs as the starting point and build from there. The SAS Studio task can be simple or very complex depending on your needs. Now that you know the basics, you can learn more about SAS Studio tasks and all of the tools that you can use from the *SAS Studio: Developer’s Guide to Writing Custom Tasks*. Have fun!

RECOMMENDED READING

- *SAS Studio: Developer’s Guide for Writing Custom Tasks*
- *SAS Studio: Getting Started Writing Your First Custom Task*

All SAS Studio documentation is available at http://support.sas.com/documentation/online/sasstudio.

CONTACT INFORMATION

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

Marie Dexter
SAS Institute Inc.
919-677-8000
marie.dexter@sas.com
www.sas.com

Kris Kiser
SAS Institute Inc.
919-677-8000
kris.kiser@sas.com
www.sas.com

Amy Peters
SAS Institute Inc.
919-677-8000
amy.peters@sas.com
www.sas.com

Christie Corcoran
SAS Institute Inc.
919-677-8000
Christie.corcoran@sas.com
www.sas.com