Paper SAS4537-2020

SAS® Studio Custom Tasks: Tips and Tricks for the Adventurous Task Author

Olivia Wright, SAS Institute Inc.

ABSTRACT

SAS® Studio provides built-in point-and-click tasks for generating and executing complex SAS® code. SAS Studio also enables users to embark on the journey of creating their own interface for their own SAS code, known as a custom task. Building a custom task is easier than you might think. There are great resources available for getting started writing custom tasks: SAS® Communities articles, GitHub examples, free e-learning, and previous SAS® Global Forum papers.

But what about those adventurous task authors who have progressed out of the "getting started" phase? This paper focuses on more daring custom task concepts that aren't covered in introductory material. Examples include writing the optional requirements and dependencies sections, creating a multi-step (multiple-task) workflow, incorporating Apache Velocity Template Language code beyond the #foreach, and working with SAS® Cloud Analytic Services (CAS) tables.

Join me on this quest to create advanced custom tasks that push the limits and incorporate the features provided by SAS Studio and Apache Velocity Template Language.

INTRODUCTION

Putting together a quick custom task is relatively simple. You may have a short SAS program that you want to build into a task to allow the user to select a different data set or variable. This kind of thing can be done easily. For most first-time task authors, everything you need to create your first task can be found in the built-in "Sample Task" or by looking at online examples. However, most users progress rather quickly out of the "getting started" phase and into more elaborate custom task work.

Once the introductory steps are mastered, questions start to arise: How can I hide or show one control based on the value of another? Can I keep the code from generating until all of the necessary options are filled out? Is it possible to have one piece of code execute, and then fill in controls of the task based on those results? How can I maximize task functionality and style? Are there any differences between task writing for SAS® 9.4 and SAS® Viya? This paper aims to answer those questions and help task authors venture into the next level of custom task development.

The full code for all example tasks in this paper is available on the Custom Task Tuesday GitHub.

OPTIONAL TASK SECTIONS

Tasks are written using the Custom Task Model, which is based on XML. Tasks are made up of six sections: Registration, Metadata, UI, Dependencies, Requirements, and Code Template. Dependencies and Requirements are both optional, meaning that you don't need to have them in order for the task to work.

Because these sections are optional, they are left out of most getting-started materials. In fact, neither dependencies nor requirements are used in the built-in "Sample Task" in SAS Studio. In the built-in "Advanced Task," the Dependencies section is used but the Requirements section is not.

The quickest way to step-up your task development is by mastering the use of these two valuable sections.

DEPENDENCIES SECTION

The Dependencies section is the first optional task section. It specifies how certain options (or controls) rely on one another in order for the task to work properly. The Dependencies section is infinitely useful because it enables you to show/hide and enable/disable one control based on the current value of another control. For example, you could have a variable selector that disappears and reappears based on the value of a check box labeled Add BY variable.

Another use of the Dependencies section is that it allows the task author to build multiple workflows into a single task. This can be accomplished by having a single radio button or combobox selection that asks the user which workflow they are interested in executing, and then using dependencies to populate the rest of the task with the necessary options based on the value of that first selection. For example, you could have a combobox or drop-down list with the options **Run Sally's Analysis** and **Run Dave's Analysis**. You could then use Dependencies to change the rest of the options available in the task based on which analysis was chosen.

To create a dependency, you specify the dependency condition, the target control that you want to change if the condition is true, and the action that you want to take on the control if the condition is true. Action options include "show," "hide," "enable," "disable," and "set."

Task Example

To show the functionality of the Dependencies section, we will look at the first example task for this paper: "Sonification Task." This task enables the user to select a data set and create a graph that is sonifiable and supported by the SAS Graphics Accelerator. In order to use this task, you must have installed the SAS Graphics Accelerator Chrome browser extension. Figure 1 shows what the task looks like.

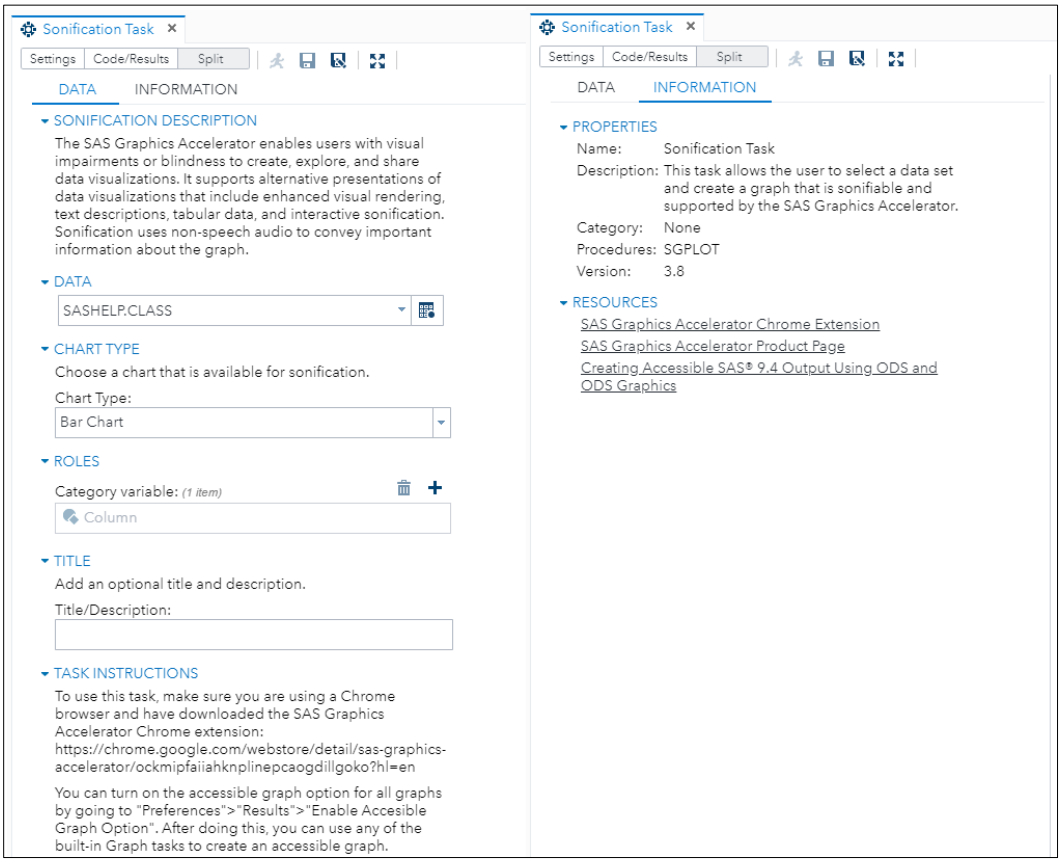

Figure 1. Sonification Task

In this example task, the Dependencies section is used to control which variable options appear for each chart type. For example, a bar chart requires only one variable (category) while a bubble plot requires three variables (x, y, and size). Here is the code for the Dependencies section:

```
<Dependencies>
   <Dependency condition="($comboTYPE == 'vbar' || $comboTYPE ==
   'hline' || $comboTYPE == 'pie')">
      <Target option="VAR" conditionResult="true"  action="show"/>
      <Target option="VAR" conditionResult="false" action="hide"/>
   </Dependency>
   <Dependency condition="($comboTYPE == 'vbox' || $comboTYPE ==
   'histogram')">
      <Target option="NVAR" conditionResult="true"  action="show" />
      <Target option="NVAR" conditionResult="false" action="hide" />
   </Dependency>
   <Dependency condition="($comboTYPE == 'bubble' || $comboTYPE ==
   'heatmap' || $comboTYPE == 'scatter')">
      <Target option="XVAR" conditionResult="true"  action="show"/>
      <Target option="XVAR" conditionResult="false" action="hide"/>
      <Target option="YVAR" conditionResult="true"  action="show"/>
      <Target option="YVAR" conditionResult="false" action="hide"/>
   </Dependency>
   <Dependency condition="($comboTYPE == 'bubble')">
      <Target option="SIZEVAR" conditionResult="true"  action="show"/>
      <Target option="SIZEVAR" conditionResult="false" action="hide"/>
   </Dependency>
</Dependencies>
```

REQUIREMENTS SECTION

The Requirements section is the second optional section. It specifies conditions for the task to run. If the condition has been met, then SAS code is generated. If the condition has not been met, the SAS code will not generate, and a message will be displayed. The Requirements section is not as widely used as the Dependencies section, but there are still times when it can come in handy.

To create a requirement, you specify the requirement condition and the message to be displayed if the requirement is not met. In most cases, it is easier to specify the condition that would fail the requirement and then negate it (by using the standard "!"), rather than specify the condition that would pass the requirement.

One final important note about the Requirements section is the timing. Requirements are always evaluated after dependencies when the task is being used. Any changes to dependencies are included in the calculation of whether requirements have been satisfied.

Task Example

Let's look at the Sonification Task again to see an example Requirements section. In this task, the requirements are used to ensure that the user has selected the necessary variables for the chosen type. For example, if the user selects bar chart from the Chart Type combobox, we want to require that the category variable array is not empty. Here is the code for the Requirements section:

```
<Requirements>
   <Requirement condition="(!(($comboTYPE == 'vbar' || $comboTYPE ==
   'hline' || $comboTYPE == 'pie') &amp;&amp; ($VAR.size() == 0)))" >
      <Message nlsKey="varSelectionMsgKey">Select a category
       variable.</Message>
   </Requirement>
   <Requirement condition="(!(($comboTYPE == 'vbox' || $comboTYPE ==
   'histogram') &amp;&amp; ($NVAR.size() == 0)))">
      <Message nlsKey="nvarSelectionMsgKey">Select an analysis
      variable.</Message>
   </Requirement>
   <Requirement condition="(!(($comboTYPE == 'bubble' || $comboTYPE ==
   'heatmap' || $comboTYPE == 'scatter') &amp;&amp; ($XVAR.size() ==
   0)))">
      <Message nlsKey="xvarSelectionMsgKey">Select an X
      variable.</Message>
   </Requirement>
   <Requirement condition="(!(($comboTYPE == 'bubble' || $comboTYPE ==
   'heatmap' || $comboTYPE == 'scatter') &amp;&amp; ($YVAR.size() ==
   0)))">
      <Message nlsKey="yvarSelectionMsgKey">Select a Y
      variable.</Message>
   </Requirement>
   <Requirement condition="(!(($comboTYPE == 'bubble') &amp;&amp;
   ($SIZEVAR.size() == 0)))">
      <Message nlsKey="sizevarSelectionMsgKey">Select a size
      variable.</Message>
   </Requirement>
</Requirements>
```

When the requirement is not met, a message ("Select a category variable") is displayed in the code window, as seen in Figure 2. After the requirement is met, the SAS code is generated as seen in Figure 3.

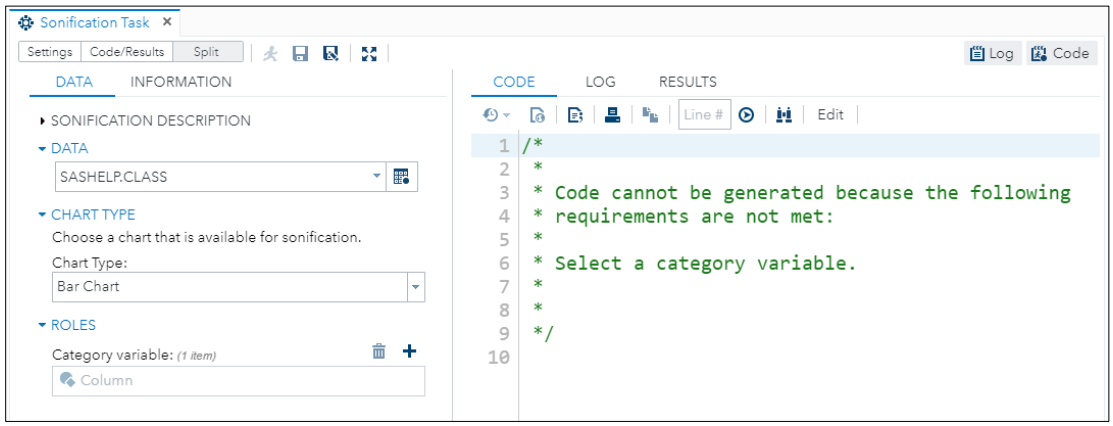

Figure 2. Requirements Not Met

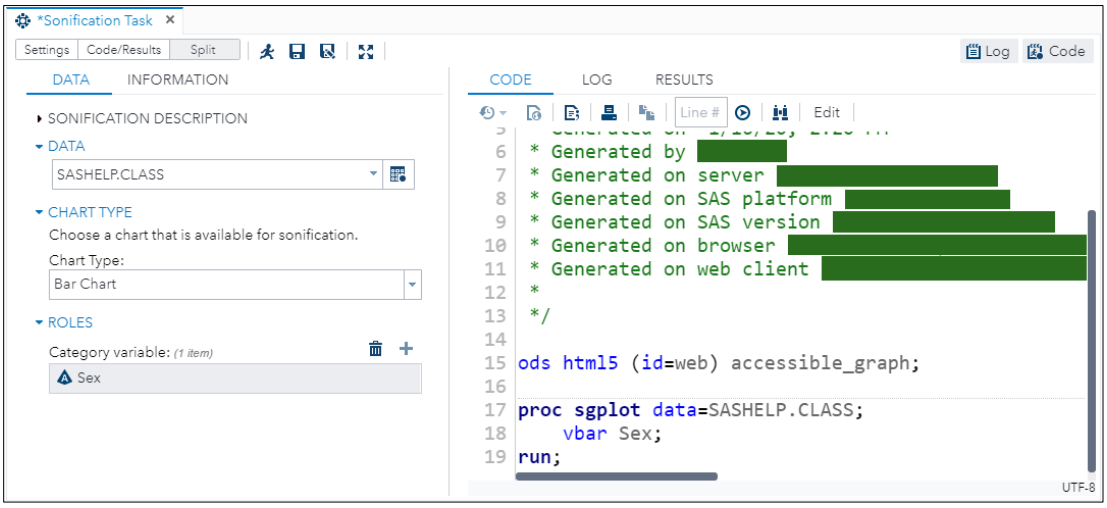

Figure 3. Requirements Met

MULTIPLE TASK WORKFLOW

Our next topic for advanced task authors is creating a multiple task workflow. Often, task authors would like to run a section of code and incorporate the results back into the task prompt. While that isn't something that is possible to accomplish within a single task, a common strategy is to create a multiple task workflow or dependent tasks labeled "Step 1," "Step 2," and so on. These separate tasks must then be run consecutively by the user. It is helpful to label tasks with the prefix "Step 1" and "Step 2" because tasks are listed in alphabetical order in their folders. This naming practice also helps emphasize to the user that running the second step will not work unless you have previously run the first step.

Task Example

For an example of a multi-task workflow, we are going to look at the following tasks: "Step 1 – Retrieve Report Images" and "Step 2 – Save Report Images." These tasks take advantage of the SAS Viya reportImages Service and use code written by Mike Drutar that can be found on his GitHub page. These tasks must be run in SAS Studio 5.2 Enterprise (or later) session within a SAS Viya 3.4 (or later) environment that contains the SAS® Visual Analytics report that is being called.

The Step 1 task (shown in Figure 4) enables you to enter a report URI for a SAS Visual Analytics report. It then retrieves images of each tab of the report and creates a table with information on each tab, as well as displays the report images in the results window.

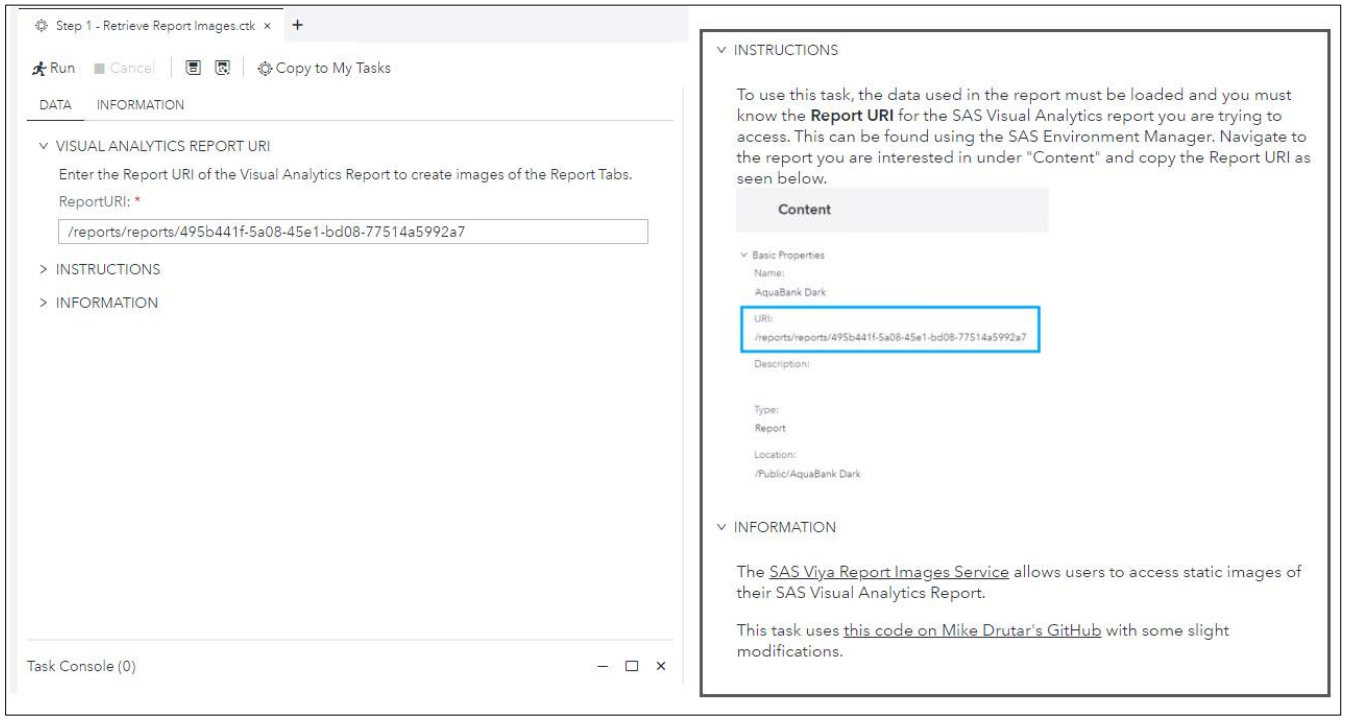

Figure 4. Step 1 – Retrieve Report Images Task

The Step 2 task (shown in Figure 5) uses the table created in the first step to allow the user to choose a report tab and download the image file to their specified location.

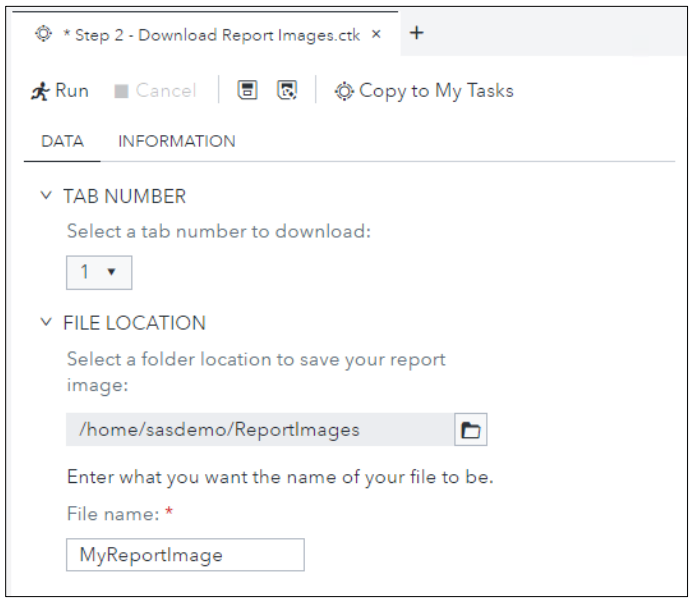

Figure 5. Step 2 – Save Report Images Task

In addition to showing the functionality of a multiple task workflow, these tasks are also examples of the new markdown object available for use in custom task development starting in SAS Studio 5.2. Markdown is a plain-text language that allows users to format their text to make it look more attractive. With Markdown, users can add text with headings, indentations, images, links, and things like bolding and italics to their tasks. The "Sample Task" available in SAS Studio 5.2 shows an example of all of the possible options in the markdown object.

Here is the code for the two markdown objects used in the "Step 1 – Retrieve Report Images".

```
<Option inputType="markdown" name="markdownText1" >
To use this task, the data used in the report must be loaded and you
must know the **Report URI** for the Visual Analytics report you are
trying to access. This can be found using the SAS Environment Manager.
Navigate to the report you are interested in under "Content" and copy
the Report URI as seen below.
![INSTRUCTIONS](https://communities.sas.com/t5/image/serverpage/image-
id/35558i00BCEA673B7223D3/image-size/large?v=1.0&amp;px=600)
</Option>
<Option inputType="markdown" name="markdownText2">
The [SAS Viya Report Images
Service](https://developer.sas.com/apis/rest/Visualization/#report-
images) allows users to access static images of their SAS Visual
Analytics Report.

This task uses [this code on Mike Drutar's
GitHub](https://github.com/sascommunities/sas-global-forum-
2019/blob/master/3156-2019-
Drutar/ViyaAPI_SASCode/create_VA_svg_image.sas) with some slight
modifications.
</Option>
```

ADVANCED VELOCITY TEMPLATE LANGUAGE

The Apache Velocity Template Language is the scripting language used for the Code Template portion of your task. Before we dive in, let's make sure we have our Velocity terminology straight.

- Variables: `$variables` are Velocity references that usually refer to a specific control in your task. All references are preceded by the "$" symbol.
- Directives: `#directives` are Velocity statements that perform some action and allow for code manipulation. These are preceded by the "#" symbol.
- Methods: `$variable.methods()` are Velocity references that refer to Java methods that perform a useful action on the variable.

If you have written a task before, you are probably familiar with a few of the more common directives and methods available in Apache Velocity Template Language. However, there is so much more functionality in Velocity for task authors to utilize.

USEFUL VELOCITY DIRECTIVES

There are many useful directives in Velocity that are necessary for task development. These allow the task author to manipulate the code that the task generates. Table 1 shows a list of helpful Velocity directives.

Directive	Description
`#if #elseif #else #end`	Allows conditional logic
`#foreach #end`	Loops through items in a list
`#set`	Sets a value for a Velocity variable (similar to a SAS %LET statement)
`#break`	Stops a loop in a `#foreach`

Table 1. Velocity Directives

For more information on Velocity directives, see the "Directives" section of the Apache documentation.

USEFUL JAVA.UTIL.LIST METHODS

Role selector variable references in the custom task model are array references in Velocity. All array references in Velocity are treated as fixed-link lists (a list of variables), and we are able to call `java.util.List` methods on those lists. The methods in Table 2 are useful for accessing variable lists.

Method	Description
`.isEmpty()`	Returns if the list is empty
`.size()`	Returns the size of the list
`.get(#)`	Gets a certain value in the list Tip: This is helpful when you have a role selector that you have restricted to allow only one variable. You can call `$var.get(0)` to get that variable without having to loop through the list

Table 2. java.util.List Methods

For more information on Velocity methods, see the "Methods" section of the Apache documentation.

PREDEFINED VELOCITY VARIABLES

The predefined Velocity variables and methods in the Table 3 were introduced to meet specific needs of task authors at SAS. These can be helpful in many situations.

Predefined Variable	Method	Description
`$CTMUtil`	`quoteString()`	Wraps a string in single quotation marks
	`doubleQuoteString()`	Wraps a string in double quotation marks
	`isProductLicensed()`	Checks to see if a specific product is installed
	`toSASName()`	Transforms a string into SAS naming conventions
`$CTMMathUtil`	`getMin()`	Returns the smallest value of an array of doubles passed in
	`getMax()`	Returns the largest value of an array of doubles passed in
	`getSum()`	Returns the sum of all the values of an array of doubles passed in

Table 3. Predefined Velocity Variables

The variables and methods in Table 3 are all detailed (with examples) in the "Common Utilities for CTM Developers" section of *SAS® Studio 5.2: Developer's Guide to Writing Custom Tasks.*

WORKING WITH CAS TABLES

For the most part, there is no difference between writing tasks for SAS 9.4 and writing tasks for SAS Viya. However, there are two important skills you will need for working with CAS

tables in your custom tasks. The first is the ability to change the library engine for the data set selector. The second is the ability to parse the two-level file name to get the library and table names.

DATA SET SELECTOR LIBRARY ENGINE

If the SAS code you are putting into a task contains SAS Viya procedures, it is a good idea to restrict your data set selector to allow only the selection of CAS tables. Alternatively, if your SAS code uses only SAS 9 procedures, you won't want users to be able to select CAS tables as input.

When you create your data set selector or data source control, use the `libraryEngineInclude=` option to restrict engines and either include or exclude CAS tables.

Task Example

For this section we will be looking at the CAS Actions task shown in Figure 6. This task uses CAS actions that allow the user to view information about their chosen data set. A CAS action sends a request to the server, invokes the action function, and returns the result. SAS Viya procedures are built from CAS actions, but you can also code in SAS Viya by writing the actions yourself using PROC CAS. The three CAS actions used are `tables.tableinfo`, `tables.columninfo`, and `simple.summary`.

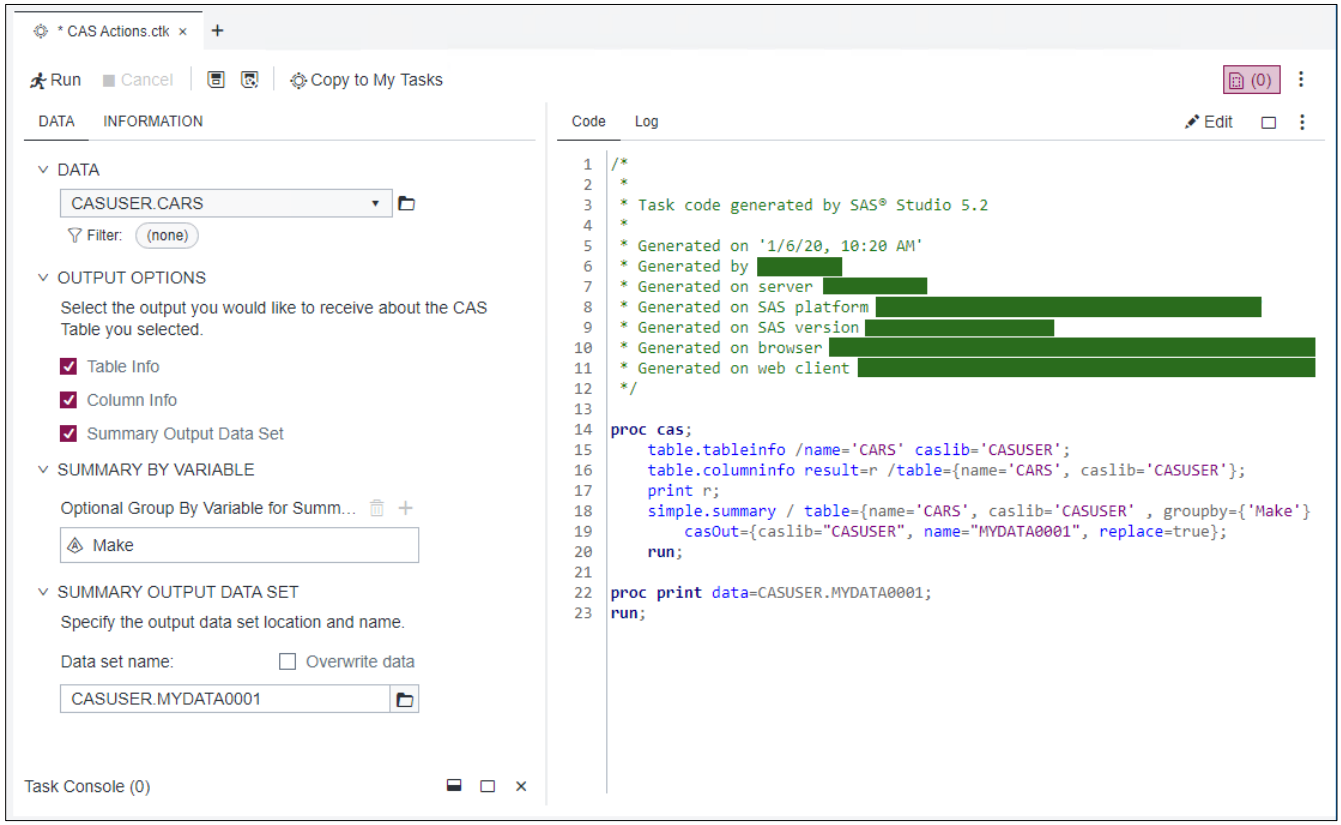

Figure 6. CAS Actions Task

The CAS Actions task will accept only a CAS table as input. The following code shows how to include only the "CAS" library engine on your data set selector:

```
<DataSources>
   <DataSource libraryEngineInclude="CAS" name="dataset" where="true">
</DataSource>
```

This results in the user seeing only CAS libraries when they make their data set selection as shown in Figure 7.

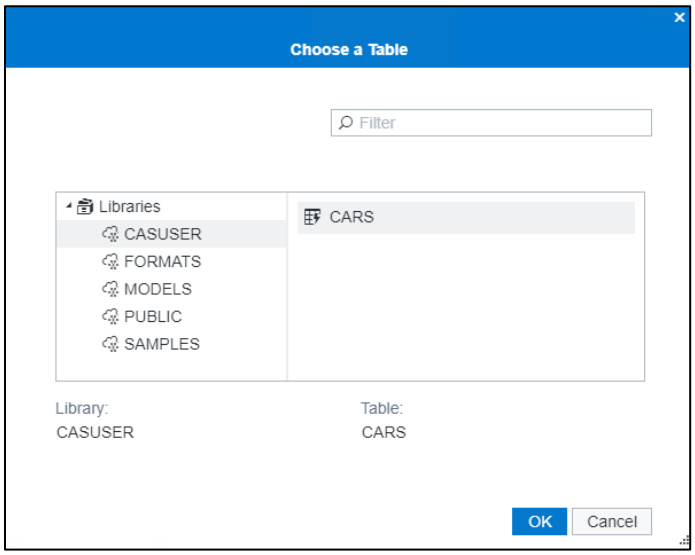

Figure 7. Data Set Selector Showing Only CAS libraries

PARSING TWO-LEVEL DATA SET NAMES

CAS actions require the library name and table name to be specified separately (using `caslib=""` and `name=""`). The data set selector provides the task author with a single string for the table in the format libraryname.datasetname, for example sashelp.class. If your SAS code incorporates CAS actions, you will need to parse this string to get separate library and table names.

Task Example

To demonstrate this, we will look again at the CAS Actions task. The following code parses the string from the data set selector to retrieve the library and data set names.

```
#set($outputLibrefIndex = $outputData.toString().indexOf("."))
#set($outputLibrefIndex1 = $outputData.toString().indexOf(".")+1)
#set($outputCASLibref = $outputData.toString().substring(0,
$outputLibrefIndex))
#set($outputCASName =
$outputData.toString().substring($outputLibrefIndex1))
#set($inputLibrefIndex = $DATASOURCE.toString().indexOf("."))
#set($inputLibrefIndex1 = $DATASOURCE.toString().indexOf(".")+1)
#set($inputCASLibref = $DATASOURCE.toString().substring(0,
$inputLibrefIndex))
#set($inputCASName =
$DATASOURCE.toString().substring($inputLibrefIndex1))
```

After you have extracted the table and libref names into Velocity macro variables ($outputCASName, $outputCASLibref, $inputCASName, and $inputCASLibref), you can reference them in your SAS Code. Here is the code from the CAS Actions task using the Velocity macro variables set above.

```
proc cas ;
#if ( $chkTABLE == 1 )
      table.tableinfo /name='$inputCASName' caslib='$inputCASLibref';
#end
#if ( $chkCOLUMN == 1 )
```

```
      table.columninfo result=r /table={name='$inputCASName',
caslib='$inputCASLibref'};
      print r;
#end
#if ( $chkSUMMARY == 1 )
simple.summary / table={name='$inputCASName', caslib='$inputCASLibref'
#if ($VAR.size() !=0 ), groupby={ #foreach( $item in $VAR ) '$item'
#end} #end}
      casOut={caslib="$outputCASLibref", name="$outputCASName",
replace=true};
#end
run;
#if ( $chkSUMMARY == 1 )
proc print data=$outputData;
run;
#end
```

CONCLUSION

Equipped with the tools described in this paper, the adventurous task author is now prepared to take their task development to the next level. By mastering the use of optional task sections, considering opportunities for multi-task workflows, incorporating advanced Velocity techniques, and appropriately using CAS tables, the custom tasks you create will be functional, elegant, and easy to use.

REFERENCES

Chrome Web Store. "SAS Graphics Accelerator." Available at https://chrome.google.com/webstore/detail/sas-graphics-accelerator/ockmipfaiiahknplinepcaogdillgoko?hl=en.

Drutar, Michael. 2020. "create_VA_svg_image.sas" Cary, NC: SAS Institute Inc. Available at https://github.com/sascommunities/sas-global-forum-2019/blob/master/3156-2019-Drutar/ViyaAPI_SASCode/create_VA_svg_image.sas

Drutar, Michael. 2019. "SAS® Viya® reportImages Service: The Report Optimization Speedometer" *Proceedings of the SAS Global Forum 2019 Conference*. Cary, NC: SAS Institute Inc. https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2019/3156-2019.pdf

SAS Institute Inc. 2019. "Common Utilities for CTM Developers" In *SAS Studio 5.2: Developer's Guide to Writing Custom Tasks*. Cary, NC: SAS Institute Inc. https://documentation.sas.com/api/collections/webeditorcdc/5.2/docsets/webeditordg/content/webeditordg.pdf.

SAS Institute Inc. 2020. SAS Product page for SAS® Graphics Accelerator. http://support.sas.com/software/products/graphics-accelerator/.

SAS Institute Inc. 2020. "Report Images" In SAS Viya REST APIs. Available at https://developer.sas.com/apis/rest/Visualization/#report-images

The Apache Software Foundation. 2020. "User Guide – Contents." In *Velocity User's Guide*. The Apache Software Foundation. https://velocity.apache.org/engine/2.0/user-guide.html

The Apache Software Foundation. 2020. "Methods." In *Velocity User's Guide*. The Apache Software Foundation. https://velocity.apache.org/engine/2.0/user-guide.html#methods

The Apache Software Foundation. 2020. "Directives." In *Velocity User's Guide*. The Apache Software Foundation. https://velocity.apache.org/engine/2.0/user-guide.html#directives

Wright, Olivia. "Custom Task Tuesday Examples." Available at https://github.com/sassoftware/sas-studio-tasks/tree/master/contributed/tasktuesdays

RECOMMENDED READING

Corcoran, Christie. Peters, Amy. 2015. "Teach Them to Fish—How to Use Tasks in SAS Studio to Enable CoWorkers to Run Your Reports Themselves" *Proceedings of the SAS Global Forum 2015 Conference*. Cary, NC: SAS Institute Inc. https://support.sas.com/resources/papers/proceedings15/SAS1831-2015.pdf

Inman, Elliot. Wright, Olivia. 2017. "Developing Your Own SAS Studio Custom Tasks for Advanced Analytics" *Proceedings of the SAS Global Forum 2017 Conference*. Cary, NC: SAS Institute Inc. https://support.sas.com/resources/papers/proceedings17/SAS0677-2017.pdf

Wright, Olivia. Matange, Sanjay. 2019. "Developing Custom SAS® Studio Tasks for Clinical Trial Graphs" *Proceedings of the PharmaSUG 2019 Conference*. Cary, NC: SAS Institute Inc. https://www.pharmasug.org/proceedings/2019/HT/PharmaSUG-2019-HT-063.pdf

ACKNOWLEDGMENTS

A special thank you to the following people for their help and contributions to this paper: Eric Bolender, Michael Drutar, Brian Gaines, Elliot Inman, Kris Kiser, Gang Meng, Ed Summers, and Qingsong Yang.

CONTACT INFORMATION

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

Olivia Wright
SAS Institute Inc.
Olivia.Wright@sas.com
Twitter: @OliviaJWright
LinkedIn: linkedin.com/in/oliviajwright/