Paper SAS-4185-2020

Creating Custom Web Applications with SAS® Viya® Jobs

Robert L. Anderson II, SAS Institute Inc., Cary, NC

ABSTRACT

In prior versions of SAS®, an oft-overlooked capability was using SAS® Stored Processes to stream HTML content and essentially create full-blown web applications that take advantage of the SAS® Platform. Using a combination of HTML, CSS, JavaScript, and SAS code, the possibilities were endless. From workflow, to data entry, to geographic mapping applications—if you could envision it, you could probably build it with SAS Stored Processes. With the introduction of SAS® Viya®, the concept of SAS Stored Processes was initially sunset. Thankfully, the SAS® Job Execution Web Application enables much of the same custom application development functionality that SAS Stored Processes did, while enabling you to leverage the full power of SAS Viya. Whether you're familiar with the old approach based on SAS Stored Processes or not, this breakout session shows you the easiest ways to leverage SAS and HTML with SAS Job Execution Web Application. The agenda for this session includes the following topics: what a SAS Viya job is; how to create and register a job in SAS® Studio; how to associate a form with a job for parameter selection; incorporating HTML into your job; and creating a full-blown web application by using jobs.

INTRODUCTION

Creating dynamic web applications requires a vast array of skills. You'll need to have some understanding of front-end development skills like JavaScript, HTML, and CSS. Depending on your type of application you might need to understand how to pull data from APIs or MySQL. An understanding of web servers like Apache and IIS is needed to serve your content to viewers. And of course, there are development frameworks like Ruby, Python, and PHP. It can all be very dizzying to a novice.

But what if you're a SAS programmer and you simply want to share insights with others via a basic application for dynamic reporting, workflow, or surfacing advanced analytics? Well, thanks to the magic of SAS® Viya® jobs, you can marry your extensive SAS expertise with a little HTML code and create dynamic web applications without all the fuss. The sky's truly the limit!

SAS Viya jobs are the SAS Viya equivalent of SAS Stored Processes in SAS®9, so if you're already comfortable with the concept of a stored procedure then SAS jobs won't be difficult to pick up. A SAS Viya job consists of a program and its definition, which includes metadata such as the job name, author, and creation date and time. At their heart, SAS jobs are simply SAS code that can run the same in a variety of web-based clients. You can use jobs for web reporting, performing analytics, building web applications, and delivering content to clients. One such client is the SAS® Job Execution Web Application. This client works with jobs that contain SAS code. These jobs can access any SAS data source or external file and create new tables, files, or other data targets that are supported by SAS.

The SAS Job Execution Web Application is a web-based client used to create, manage, and execute jobs. This application, written in Java, provides access to data in combination with a

powerful array of analysis and presentation procedures running on a server. No SAS software is required on the client machine.

To access and analyze data, a web user typically completes an HTML input form displayed by the SAS Job Execution Web Application. When the user selects the option to submit the information, data that is specified in the form is passed to a waiting SAS session as global macro variables. The SAS program runs, and the results are returned to the web browser.

While the SAS Job Execution Web Application is the engine that runs your job, beginning in SAS® Studio 5.2, you can create and define your jobs in SAS Studio. This paper covers that approach, since the SAS Job Execution Web Application might be deprecated and is already well documented.

WHAT YOU'LL NEED TO KNOW BEFORE CONTINUING

It is required that you have a solid understanding of SAS coding fundamentals. This includes the DATA step, PROC SQL, Output Delivery System, and macros. Prior knowledge of SAS Stored Processes will help but isn't required. In addition, a basic understanding of HTML is necessary as most of the code in this paper generates interactive web pages. Ancillary web development skills like CSS and JavaScript aren't essential but will help you develop more aesthetically pleasing and interactive applications.

THE BASICS OF CREATING A JOB

SAS Studio 5.2 is the optimal application for creating SAS Viya jobs. When you create a job, the %JESBEGIN utility macro sets up the job execution environment and executes before your SAS code. You use job input parameter values to control the macro. For example, specify _OUTPUT_TYPE=ods_html5 if your job uses ODS to create HTML output. The macro assigns a FILEREF named _WEBOUT to return output to the web browser or client application and issues an ODS HTML5 statement.

Scenario	_Output_Type
Job contains only SAS procedure output	ODS_HTML5
Job output contains HTML only	HTML
Job contains a mix of SAS PROCs and HTML	HTML

Table 1. When to Use Which _Output_Type Value

The following example creates a simple job that uses DATA step code to return HTML to the client. The FILENAME statement shown in Assigning a FILEREF for HTML Output is used.

```
http://host:port/SASStudioV
```

Open SAS Studio and choose New -> Job -> Definition to begin.

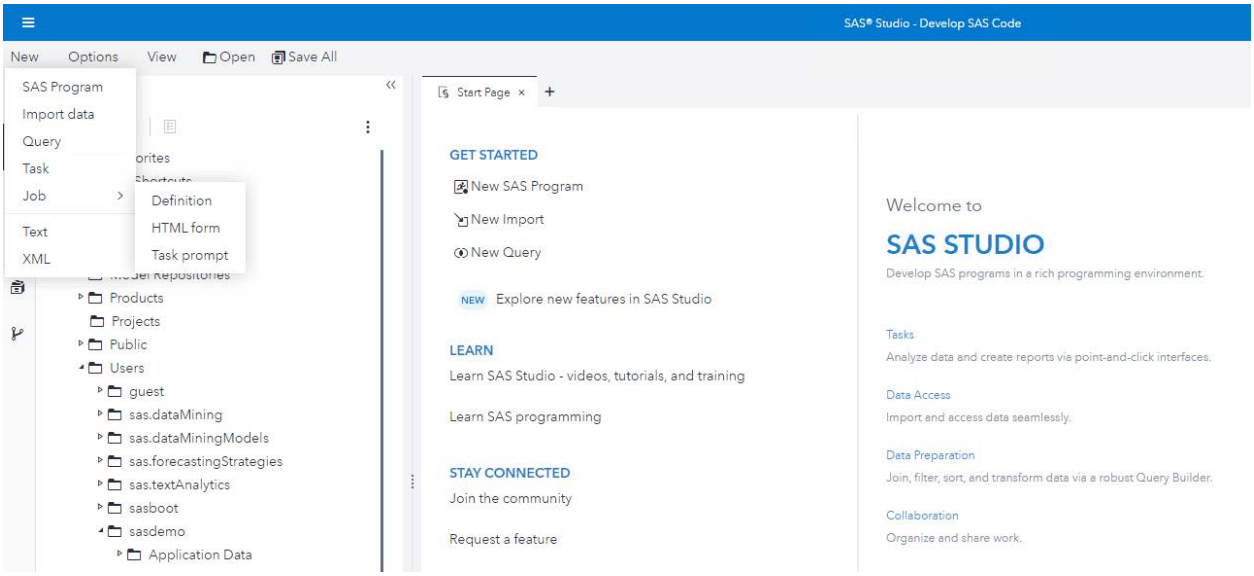


Figure 1. Creating a New Job Definition in SAS Studio

The Job Definition pane looks like the standard SAS Code window, with a couple of important exceptions. From the options menu on the far right, you can choose a form (if desired) to execute the job, view its properties, and define parameters to customize the job's output.

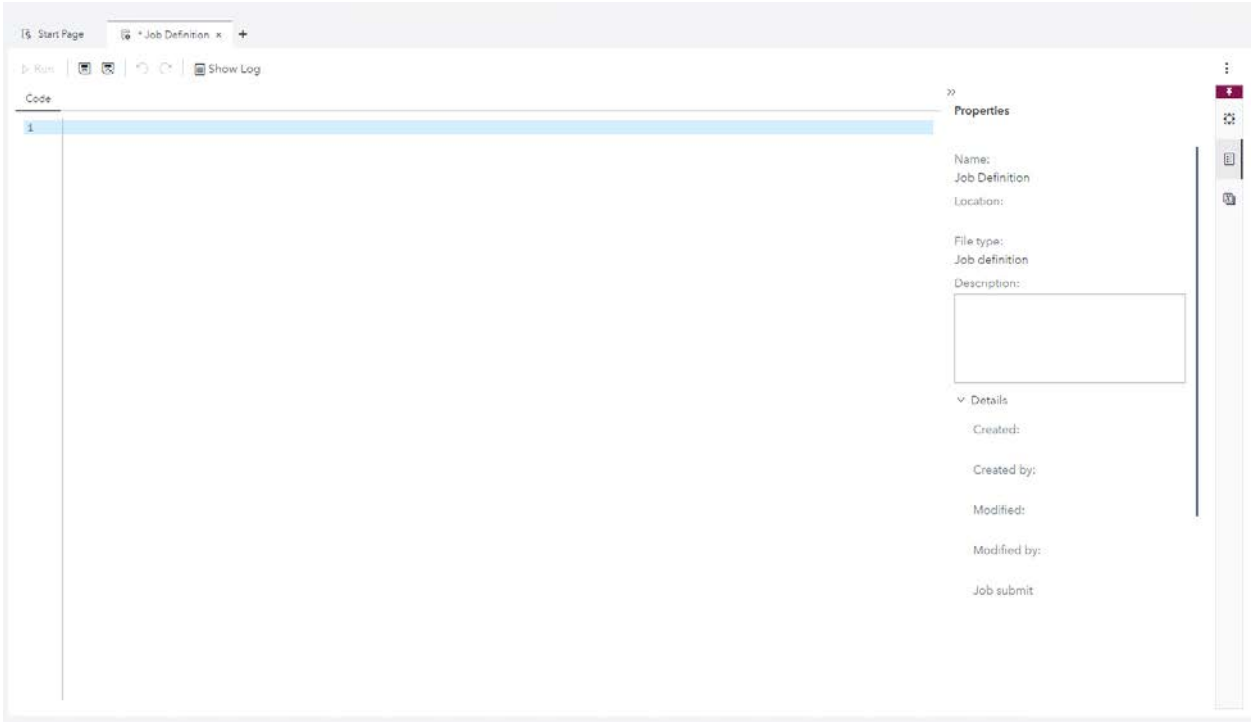


Figure 2. The Job Definition Canvas

CREATING A SIMPLE JOB THAT STREAMS SAS OUTPUT ONLY

Among the most basic uses of a SAS job is the ability to run code in a non-interactive manner to return SAS procedure output to the browser. The following example creates a simple job that uses a PROC PRINT on the SASHELP.CARS table and streams output to the client. There's

little more to this approach than creating a SAS job with your code and specifying the parameter _Output_Type = ods_html5:

```
proc print data=SASHELP.CARS (obs=10);
run;
```

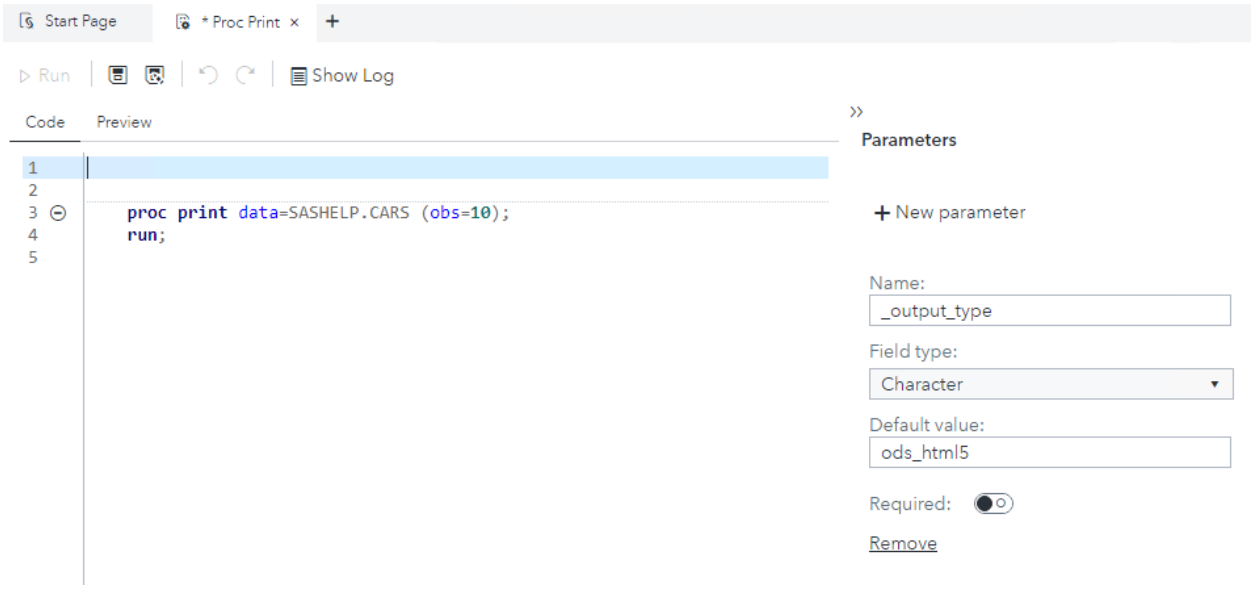


Figure 3. Job Properties

Save the job and execute it by pressing the Run button. The following output will be displayed in the job's Preview pane.

Obs	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight	Wheelbase	Length
1	Acura	MDX	SUV	Asia	All	$36,945	$33,337	3.5	6	265	17	23	4451	106	189
2	Acura	RSX Type S 2dr	Sedan	Asia	Front	$23,820	$21,761	2.0	4	200	24	31	2778	101	172
3	Acura	TSX 4dr	Sedan	Asia	Front	$26,990	$24,647	2.4	4	200	22	29	3230	105	183
4	Acura	TL 4dr	Sedan	Asia	Front	$33,195	$30,299	3.2	6	270	20	28	3575	108	186
5	Acura	3.5 RL 4dr	Sedan	Asia	Front	$43,755	$39,014	3.5	6	225	18	24	3880	115	197
6	Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	Front	$46,100	$41,100	3.5	6	225	18	24	3893	115	197
7	Acura	NSX coupe 2dr manual S	Sports	Asia	Rear	$89,765	$79,978	3.2	6	290	17	24	3153	100	174
8	Audi	A4 1.8T 4dr	Sedan	Europe	Front	$25,940	$23,508	1.8	4	170	22	31	3252	104	179
9	Audi	A41.8T convertible 2dr	Sedan	Europe	Front	$35,940	$32,506	1.8	4	170	23	30	3638	105	180
10	Audi	A4 3.0 4dr	Sedan	Europe	Front	$31,840	$28,846	3.0	6	220	20	28	3462	104	179

Figure 4. Job Output

CREATING A SIMPLE JOB THAT STREAMS HTML ONLY

The following example creates a simple job that uses DATA step code to return HTML to the client. If you're familiar with this approach from creating streaming Stored Processes, you'll recall the need to wrap your HTML in PUT statements and single quotation marks. This is easily illustrated to display output for the classic Hello World message. Unlike the SAS PROC

example, our output requires the parameter _output_type = html in order to properly stream the results:

```
data _null_;
file _webout;
      put '<head><title>Hello World!</title></head>';
      put '<body>';
      put '<h1>Hello World!</h1>';
      put '</body>';
      put '</html>';
      put '<html>';
run;
```

The output consists entirely of HTML generated by SAS and streamed via the Job Execution service.

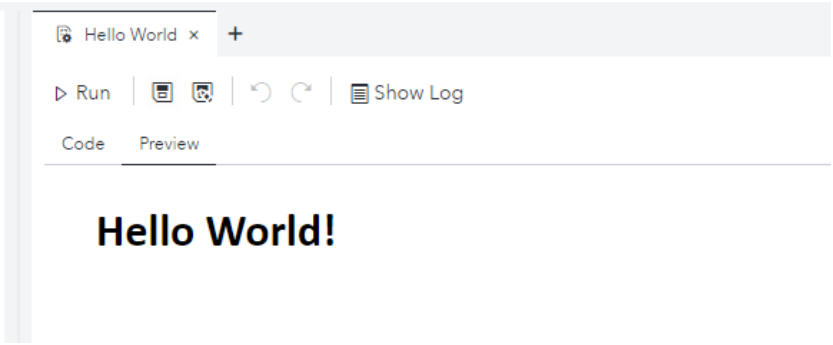


Figure 5. Job Output

CREATING A SIMPLE JOB THAT STREAMS HTML ONLY (WITHOUT PUT STATEMENTS OR QUOTATION MARKS)

While the previous approach to streaming HTML worked just fine, wrapping your HTML code in PUT statements and quotation marks could become tedious as your code grows. Luckily, by wrapping your code into an INFILE statement, the same data _null_ technique above enables you to skip that step. The output still needs to be set to _output_type = html in order for this approach to work:

```
data _null_;
  format infile $char256.;
  input;
  infile = resolve(_infile_);
  file _webout;
  put infile;
cards4;
<html>
<head>
<body>
<p>Hello World, Isn't This Just Sooooo Much Easier?!?<p>
</body>
</html>
;;;;
run;
```

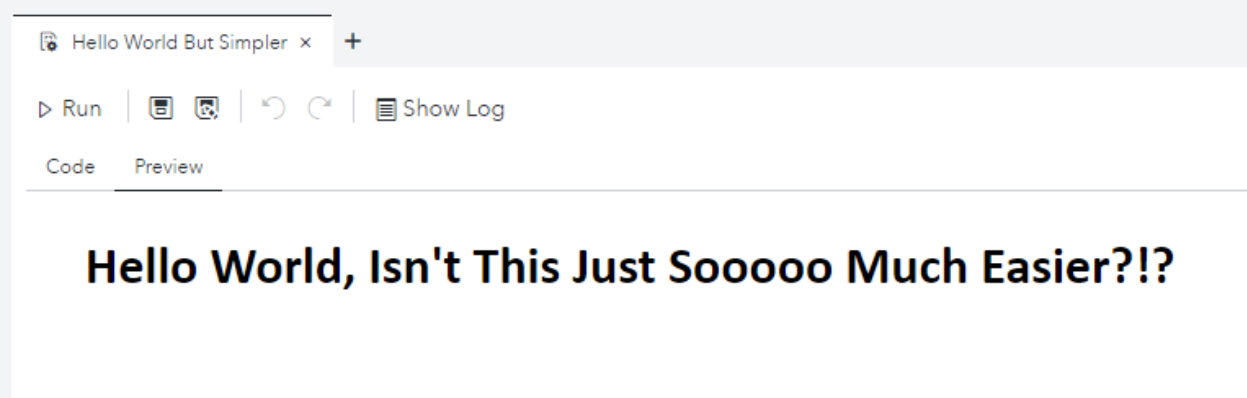


Figure 6. Job Output

It helps to understand both approaches to fully develop your applications. You'll typically use this second method when you're simply cutting and pasting standard HTML to create an interface, but certain instances will require the PUT method to incorporate dynamic URLs that are generated by SAS. We'll draw these distinctions as we proceed.

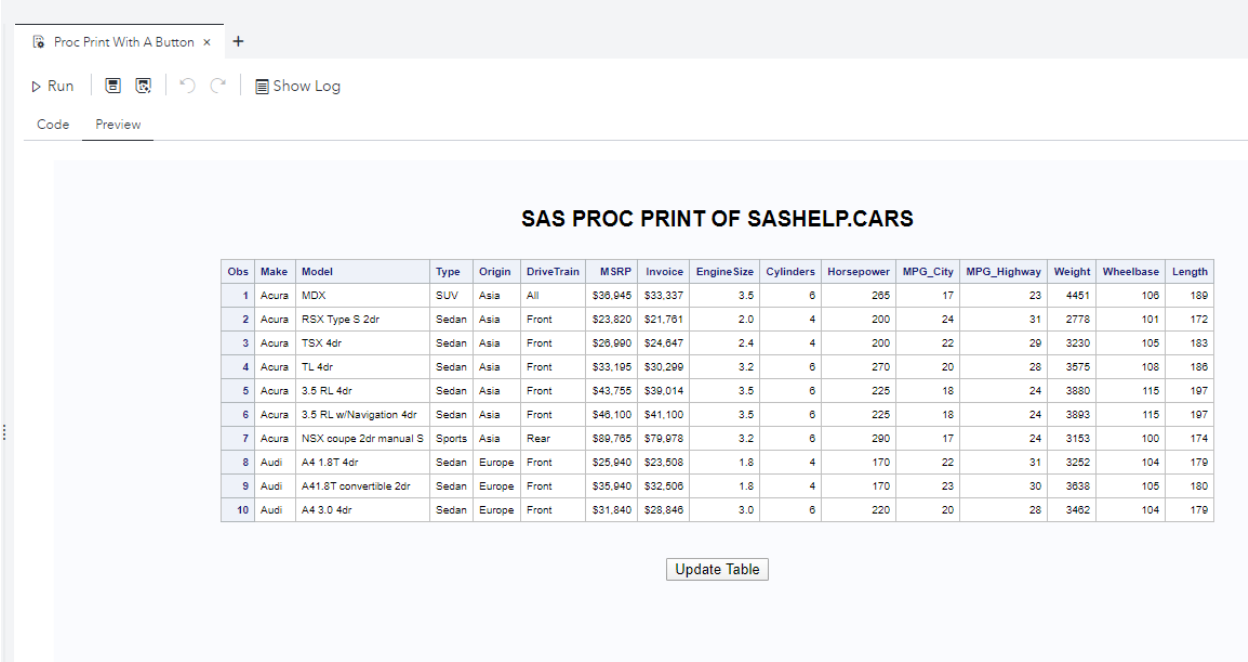
CREATING SIMPLE JOB STREAMS (BOTH HTML AND SAS PROC OUTPUT)

We can combine both of our previous techniques to create output that includes both SAS PROC output *and* custom HTML. This requires us to toggle between output types programmatically. In the job definition's parameters, specify `_output_type = html` in order to render the HTML and capture SAS PROC code that creates output within ODS HTML tags. SAS procedure output has to be explicitly captured between ODS HTML tags in order to render. This approach allows us to freely switch back and forth between both output types, giving us the foundation for any application development.

```
data _null_;
  format infile $char256.;
  input;
  infile = resolve(_infile_);
  file _webout;
  put infile;
cards4;
<html>
<head>
<body>
<center>
<br><br>
<h1>SAS PROC PRINT OF SASHELP.CARS</h1>
</center>
<br>
</body>
</html>
;;;;
run;

ods html file=_webout;

options nodate nonumber;
proc print data=sashelp.cars (obs=10);run;
```

```
ods html close;

…more html in data _null_ to display button, etc.
```

The output looks like this.



Obs	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight	Wheelbase	Length
1	Acura	MDX	SUV	Asia	All	$36,945	$33,337	3.5	6	265	17	23	4451	106	189
2	Acura	RSX Type S 2dr	Sedan	Asia	Front	$23,820	$21,761	2.0	4	200	24	31	2778	101	172
3	Acura	TSX 4dr	Sedan	Asia	Front	$26,990	$24,647	2.4	4	200	22	29	3230	105	183
4	Acura	TL 4dr	Sedan	Asia	Front	$33,195	$30,299	3.2	6	270	20	28	3575	108	186
5	Acura	3.5 RL 4dr	Sedan	Asia	Front	$43,755	$39,014	3.5	6	225	18	24	3880	115	197
6	Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	Front	$46,100	$41,100	3.5	6	225	18	24	3893	115	197
7	Acura	NSX coupe 2dr manual S	Sports	Asia	Rear	$89,765	$79,978	3.2	6	290	17	24	3153	100	174
8	Audi	A4 1.8T 4dr	Sedan	Europe	Front	$25,940	$23,508	1.8	4	170	22	31	3252	104	179
9	Audi	A41.8T convertible 2dr	Sedan	Europe	Front	$35,940	$32,506	1.8	4	170	23	30	3638	105	180
10	Audi	A4 3.0 4dr	Sedan	Europe	Front	$31,840	$28,846	3.0	6	220	20	28	3462	104	179

Figure 7. Job Output

ASSOCIATING AN HTML FORM WITH YOUR JOB TO MAKE IT DYNAMIC

At the core of most dynamic applications, a user selects some parameters and then has those parameters run against code to return the desired output. While it's entirely possible for your parameter selection screen (aka: a form) to be another SAS job, if the parameter screen is simple, we can just add the HTML file by associating it with the job definition.

STEP ONE – CREATE THE JOB

We can make our simple PROC PRINT from the prior example dynamic by allowing users to choose a vehicle type and max price. We'll take this code and make a couple of changes to accept parameters from the form. To ensure that everything resolves correctly, we'll define our parameter variables as global.

```
%global VEHICLE_TYPE MSRP_MAX;

proc print data=SASHELP.CARS;
where type = "&VEHICLE_TYPE" and MSRP < &MSRP_MAX;
run;
```

Before associating this job with a form, we'll need to test to ensure it works first, so let's specify default values for the two variables and define them as parameters. Since our output consists solely of SAS PROC PRINT, we'll specify `_output_type = ods_html5`.

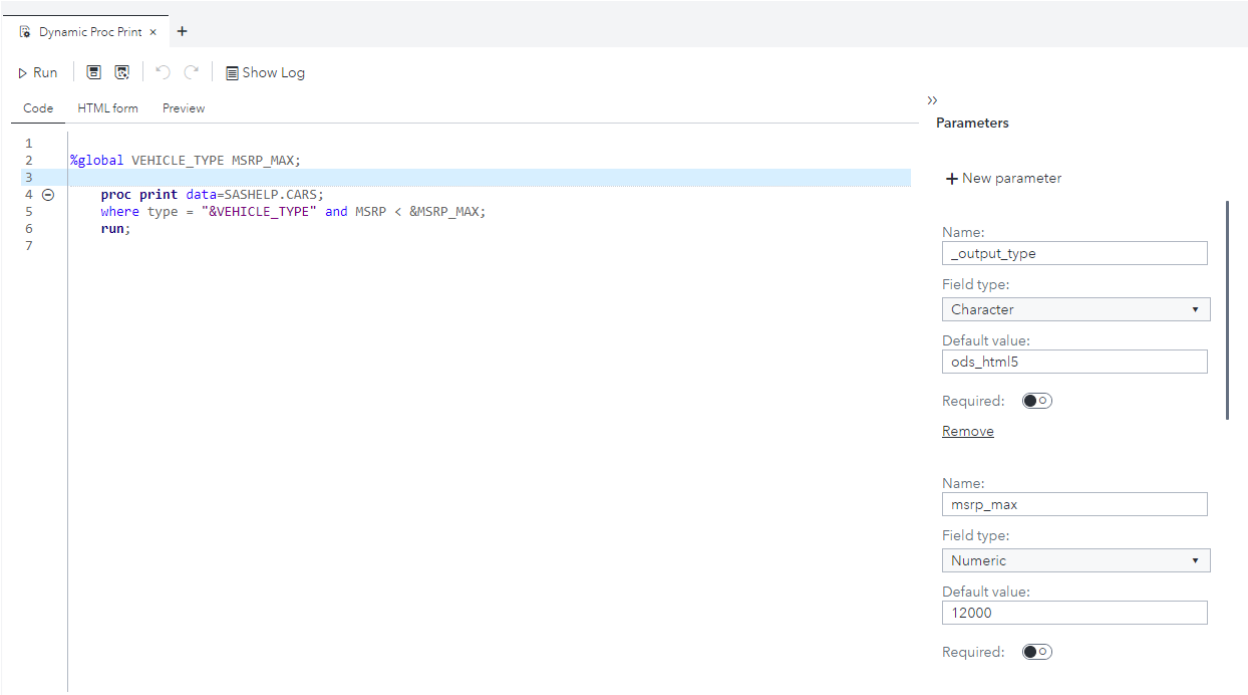


Figure 8. Job Properties

As this point, if we submit our job, it'll accept the parameters defined in its properties and generate the desired output: a list of sedans with an MSRP of less than $12,000.

Obs	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight	Wheelbase	Length
68	Chevrolet	Aveo 4dr	Sedan	USA	Front	$11,690	$10,965	1.6	4	103	28	34	2370	98	167
169	Hyundai	Accent 2dr hatch	Sedan	Asia	Front	$10,539	$10,107	1.6	4	103	29	33	2255	96	167
170	Hyundai	Accent GL 4dr	Sedan	Asia	Front	$11,839	$11,116	1.6	4	103	29	33	2290	96	167
171	Hyundai	Accent GT 2dr hatch	Sedan	Asia	Front	$11,939	$11,209	1.6	4	103	29	33	2339	96	167
207	Kia	Rio 4dr manual	Sedan	Asia	Front	$10,280	$9,875	1.6	4	104	26	33	2403	95	167
208	Kia	Rio 4dr auto	Sedan	Asia	Front	$11,155	$10,705	1.6	4	104	25	32	2458	95	167
346	Saturn	Ion1 4dr	Sedan	USA	Front	$10,995	$10,319	2.2	4	140	26	35	2692	103	185
383	Toyota	Echo 2dr manual	Sedan	Asia	Front	$10,760	$10,144	1.5	4	108	35	43	2035	93	163
384	Toyota	Echo 2dr auto	Sedan	Asia	Front	$11,560	$10,896	1.5	4	108	33	39	2085	93	163
385	Toyota	Echo 4dr	Sedan	Asia	Front	$11,290	$10,642	1.5	4	108	35	43	2055	93	163

Figure 9. Job Output

While this is helpful, it would require users to modify the job's parameter properties each time they want to see their desired output, which isn't feasible. Thankfully, you can easily associate a form with this job to display an HTML page, enabling users to choose the values they want.

STEP TWO – CREATE THE FORM

On the job definition's Associate a Form options pane, choose an HTML form. A new HTML form tab will open beside the job's Code tab. This is where you'll add the HTML "form" that will call your job and make it dynamic. Since this is standard HTML, there are no data _null_ sections. You can create your HTML in any editor you'd like. You can choose any HTML objects you'd like to select the values, but be certain to use the same SAS macro variable names in the job's code. In this case, those are MSRP_MAX (for the maximum cost of the car) and VEHICLE_TYPE (to choose the desired car type).

```
<!DOCTYPE html>
<head>
<title>Simple HTML Example</title>
</head>
<body>

<h1>Select Parameters To Send To SAS Proc Print</h1>

<form action="/SASJobExecution/" target="_self">
  <input    type="hidden"    name="_program"    value="/Users/sasdemo/SGF
Examples/Dynamic Proc Print">
  <input type="hidden" name="_action"  value="execute">

<label>What's the Most You Want To Pay? </label><input type="number"
name="MSRP_MAX" min="11000" max="100000">
<br><br>
<label>What Kind Of Car Do You Want? </label>
<select name="VEHICLE_TYPE">
  <option value="Sedan">Sedan</option>
  <option value="Sports">Sports</option>
  <option value="SUV">SUV</option>
</select>

<br/>
<br/>

<input type="submit" value="Show Me My New Car">
</form>
</body>
</html>
```

You'll note a couple of important items here. Always specify */SASJobExecution/* in the *ACTION* attribute of the *FORM* tag to indicate that the form data is submitted to the SAS Job Execution Web Application for processing. You can specify _blank in the TARGET attribute to force the output to always appear in a new browser window or tab, or _self to keep it in the same screen.

The first input tag specifies that a non-visual object named _PROGRAM has a value of */SomeFolder/Simple HTML*. This value is displayed in the job's Properties dialog box as Location. This input tag indicates the location and name of the program to execute. The value that you specify is case-sensitive.

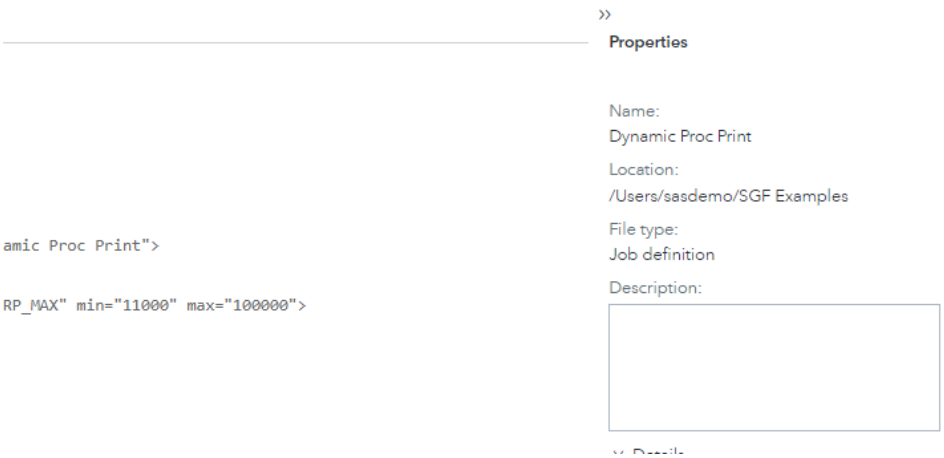


Figure 10. Job Name and Location

The second input tag specifies a value of execute for the _ACTION parameter, which overrides the default value specified in the job parameter and executes the job.

STEP THREE EXECUTING THE JOB (AND GETTING THE FORM!)

When you select Run the Job now, instead of seeing the PROC PRINT output with the default parameters specified in the Properties, we'll see the custom HTML form we just defined in the Preview pane.

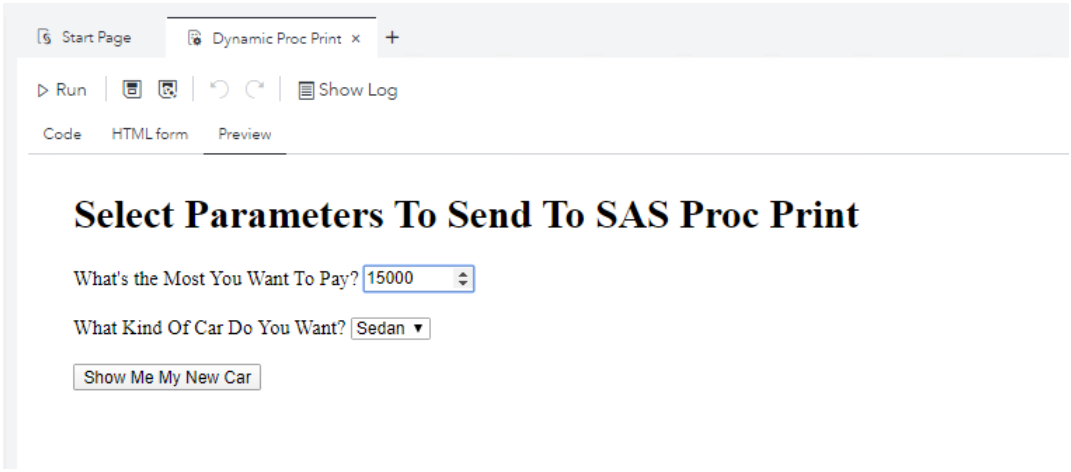
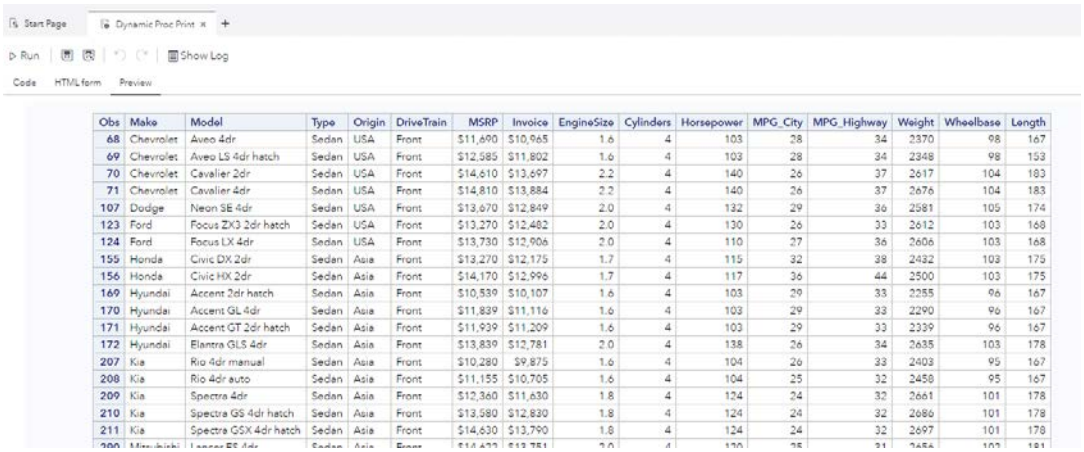


Figure 11. Parameter Input Form

Fill in values for MSRP and Car Type, and a dynamic report will be displayed.



Obs	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight	Wheelbase	Length
68	Chevrolet	Aveo 4dr	Sedan	USA	Front	$11,690	$10,965	1.6	4	103	28	34	2370	98	167
69	Chevrolet	Aveo LS 4dr hatch	Sedan	USA	Front	$12,585	$11,802	1.6	4	103	28	34	2348	98	153
70	Chevrolet	Cavalier 2dr	Sedan	USA	Front	$14,610	$13,697	2.2	4	140	26	37	2617	104	183
71	Chevrolet	Cavalier 4dr	Sedan	USA	Front	$14,810	$13,884	2.2	4	140	26	37	2676	104	183
107	Dodge	Neon SE 4dr	Sedan	USA	Front	$13,670	$12,849	2.0	4	132	29	36	2581	105	174
123	Ford	Focus ZX3 2dr hatch	Sedan	USA	Front	$13,270	$12,482	2.0	4	130	26	33	2612	103	168
124	Ford	Focus LX 4dr	Sedan	USA	Front	$13,730	$12,906	2.0	4	110	27	36	2606	103	168
155	Honda	Civic DX 2dr	Sedan	Asia	Front	$13,270	$12,175	1.7	4	115	32	38	2432	103	175
156	Honda	Civic HX 2dr	Sedan	Asia	Front	$14,170	$12,996	1.7	4	117	36	44	2500	103	175
169	Hyundai	Accent 2dr hatch	Sedan	Asia	Front	$10,539	$10,107	1.6	4	103	29	33	2255	96	167
170	Hyundai	Accent GL 4dr	Sedan	Asia	Front	$11,839	$11,116	1.6	4	103	29	33	2290	96	167
171	Hyundai	Accent GT 2dr hatch	Sedan	Asia	Front	$11,939	$11,209	1.6	4	103	29	33	2339	96	167
172	Hyundai	Elantra GLS 4dr	Sedan	Asia	Front	$13,839	$12,781	2.0	4	138	26	34	2635	103	178
207	Kia	Rio 4dr manual	Sedan	Asia	Front	$10,280	$9,875	1.6	4	104	26	33	2403	95	167
208	Kia	Rio 4dr auto	Sedan	Asia	Front	$11,155	$10,705	1.6	4	104	25	32	2458	95	167
209	Kia	Spectra 4dr	Sedan	Asia	Front	$12,360	$11,630	1.8	4	124	24	32	2661	101	178
210	Kia	Spectra GS 4dr hatch	Sedan	Asia	Front	$13,580	$12,830	1.8	4	124	24	32	2686	101	178
211	Kia	Spectra GSX 4dr hatch	Sedan	Asia	Front	$14,630	$13,790	1.8	4	124	24	32	2697	101	178

Figure 12. Dynamic Job Output After Parameters Are Submitted

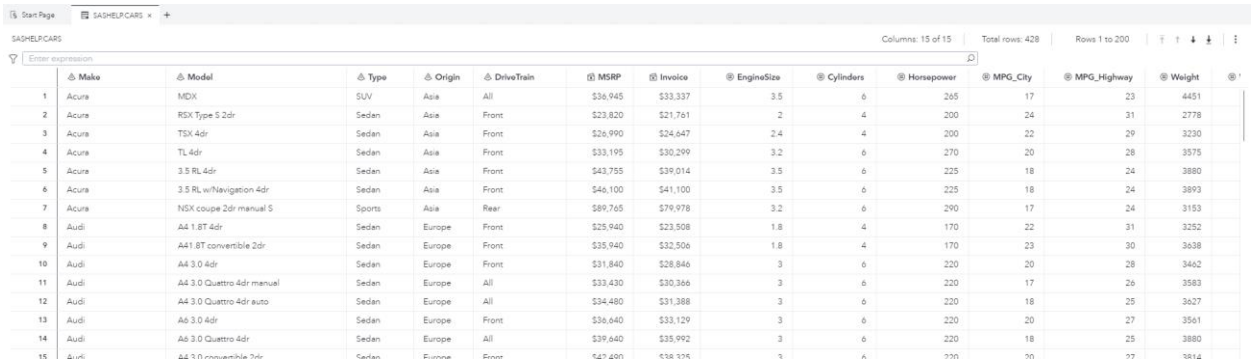
CREATING A CUSTOM FORM JOB THAT SENDS PARAMETERS TO ANOTHER JOB

While HTML forms are a good way to parameterize your SAS Viya jobs, the full power of SAS Job Execution Web Application is unleashed when you build jobs that call other jobs (which call other jobs). This technique is called *chaining* and enables you to truly build a self-contained application with SAS Viya jobs.

We'll build on the prior concepts in this paper and create an interactive application that enables users to search for a car from the SASHELP.CARS data set. We'll then display a list of cars that fit their search criteria, view a listing of the desired car available at local dealerships, and calculate monthly payments based on the car's total cost of ownership. While this sounds like a lot (and it is!) it'll give you a great understanding of what sorts of applications that you can create with a little SAS code, some HTML, and SAS Viya jobs.

STEP ONE – CREATE THE CAR SEARCH OPTIONS SCREEN

A typical starting point for a typical web-based application is allowing users to choose parameters and to bring back a subset of a table based on their selections. The SASHELP.CARS table provides an array of information about cars, trucks, sedans, and other vehicles available in the 00's.



	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight
1	Acura	MDX	SUV	Asia	All	$36,945	$33,337	3.5	6	265	17	23	4451
2	Acura	RSX Type S 2dr	Sedan	Asia	Front	$23,820	$21,761	2	4	200	24	31	2778
3	Acura	TSX 4dr	Sedan	Asia	Front	$26,990	$24,647	2.4	4	200	22	29	3230
4	Acura	TL 4dr	Sedan	Asia	Front	$33,195	$30,299	3.2	6	270	20	28	3575
5	Acura	3.5 RL 4dr	Sedan	Asia	Front	$43,755	$39,014	3.5	6	225	18	24	3880
6	Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	Front	$46,100	$41,100	3.5	6	225	18	24	3893
7	Acura	NSX coupe 2dr manual S	Sports	Asia	Rear	$89,765	$79,978	3.2	6	290	17	24	3153
8	Audi	A4 1.8T 4dr	Sedan	Europe	Front	$25,940	$23,508	1.8	4	170	22	31	3252
9	Audi	A4 1.8T convertible 2dr	Sedan	Europe	Front	$35,940	$32,506	1.8	4	170	23	30	3638
10	Audi	A4 3.0 4dr	Sedan	Europe	Front	$31,840	$28,846	3	6	220	20	28	3462
11	Audi	A4 3.0 Quattro 4dr manual	Sedan	Europe	All	$33,430	$30,366	3	6	220	17	26	3583
12	Audi	A4 3.0 Quattro 4dr auto	Sedan	Europe	All	$34,480	$31,388	3	6	220	18	25	3627
13	Audi	A6 3.0 4dr	Sedan	Europe	Front	$36,640	$33,129	3	6	220	20	27	3561
14	Audi	A6 3.0 Quattro 4dr	Sedan	Europe	All	$39,640	$35,992	3	6	220	18	25	3880

Figure 13. SASHELP.CARS Table

Many of the cars (RIP Mercury Sable!) lines and models were victims of the 2008 Automobile Industry Crisis and are no longer manufactured, but the table is still chock full of plenty of usable information. Our goal is to use this table as the basis of an application for perspective car shoppers much like Cars.com.

At a high level, most people shopping for cars have a rough idea of the sort of car that they'd like to buy (sedan, SUV, truck) and how much they want to pay for said car. Other items such as fuel economy and engine type are also important, so we'll make these four items the basis of our initial search screen.

With a few exceptions, the form that we create to accept a user's search parameters is going to consist entirely of stock HTML code. You can choose any HTML editor to create this page, but it needs to have five basic elements: four objects for selecting our parameters (car type, max price, minimum fuel efficiency, and number of cylinders) and one button to send these parameters to the next job.

Your code needs to be created as a job definition in SAS Studio. Add the parameter _output_type = html as shown in the prior examples since we're streaming only HTML, not SAS PROC output, back to the browser.

To display your HTML, you'll need to simply cut and paste it between the wrapper code below. The file _webout tells SAS Job Execution Web Application to send the HTML back to the browser:

```
data _null_;
  format infile $char256.;
  input;
  infile = resolve(_infile_);
  file _webout;
  put infile;
  cards4;
[Your HTML Code Goes Here]
;;;;
run;
```

Do not modify the wrapper code. The four semicolons are also essential. Just add your code between the lines. At this time, it's *not* possible to add references to external .CSS and JavaScript files to your code. You'll simply need to include those items directly in your program.

Test the code by submitting its job definition to confirm that your output looks as expected.

In order to "chain" this stored process to another, we'll need to add a few more items to make it fully operational. You've hopefully wrapped all of your select items and button inside of <form> tags. Modify the opening <form> tag to specify an action that takes place when the submit button is pressed. SAS Job Execution Web Application is how we'll process any code, so we'll note that for the action= value. The target= is optional, but since we want everything to flow within the same window, we'll make this value _self.

```
<form action="/SASJobExecution/" target="_self">
```

We need to now specify the name of the _program that should run when we submit our form. This file is essentially step two in our process and hasn't yet been created, but we'll add it as we go along in the value= place. Remember, this value can be obtained from the job's Properties tab as the value Location.

```
  <input type="hidden" name="_program" value /Users/sasdemo/SGF
Examples/Custom App Step 2">
```

There are many ways to create a button in HTML, but since this is all part of a form that's being submitted, we'll add the following just before we close our </form> tag.

```
<INPUT TYPE="SUBMIT" VALUE="Find My New Car">
```

These three steps bring our static HTML to life. We specify the *action* that will execute our code, specify the *value* of that *_program* in the metadata, and then *submit* the code with a button.

Everyone's code might look a bit different, but you can use the example below, making obvious edits for the _program value.

```
data _null_;
  format infile $char256.;
  input;
  infile = resolve(_infile_);
  file _webout;
  put infile;
cards4;
<HTML>
<head>
```

…style info in header

```
<BODY>

<center><font face=&font_value color="#5C5C5C" size=5>
<b>What Type Of Car Do You Want?</b></center></font>
<br>
<div class="center">
<form action="/SASJobExecution/" target="_self">
  <input type="hidden" name="_program" value="/Users/sasdemo/My
Folder/SGF Examples/6. Complex Form Job (Part Two)">
```

….select objects and HTML body

```
<INPUT TYPE="hidden" name="_debug" VALUE="131">*/
<INPUT TYPE="SUBMIT" VALUE="Find My New Car">

</center>
```

```
</FORM>
</div>
</BODY>
</HTML>
;;;;
run;
```

Your output is displayed in the Preview pane and should look like this.

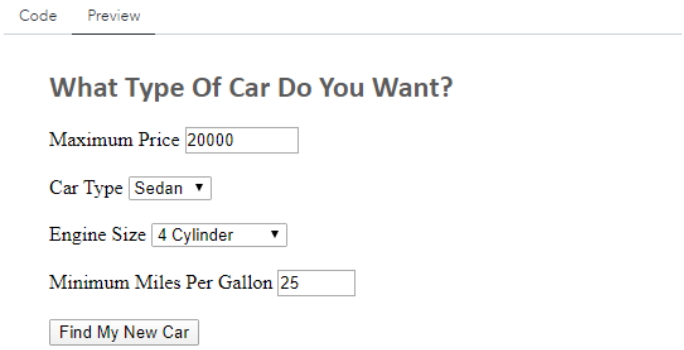


Figure 14. Car Search Parameter Selection Screen

STEP TWO – DISPLAY A LIST OF CARS THAT FIT THE USER'S CRITERIA

While our first step simply entailed allowing users to select the criteria for their car search, the magic happens when those parameters are sent to step two. Here, the parameters for Car Type, Max Price, Fuel Efficiency, and Engine Size are accepted as name-value pairs, converted to SAS macro variable values, and used to form a WHERE clause that runs against SASHELP.CARS and returns a list of cars that fit the user's criteria.

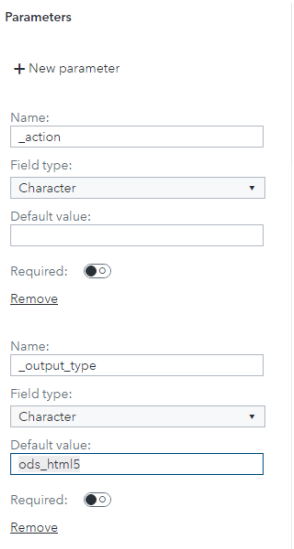


Figure 15. Job Parameter Options

Create this new job definition with the parameter _output_type = ods_html5 since we are displaying only SAS PROC output. In addition, since we'll be defining the _action type in our dynamically generated URL, remove the default value for the _action parameter.

In addition to displaying a list of cars, we'll want to expand the application by linking it to a list of local dealerships that see the desired car. This will be displayed as a hyperlink in the output table and we'll need to construct it in two steps. First, we'll form the _program portion of the URL and assign it to a macro variable called INFO_URL. Create the next job definition as a placeholder, and then cut and paste the Job submit: property from your job definition, beginning with the ?_program= portion. Add &_action=wait, which tells SAS Job Execution Web Application to run the program when the URL is submitted.

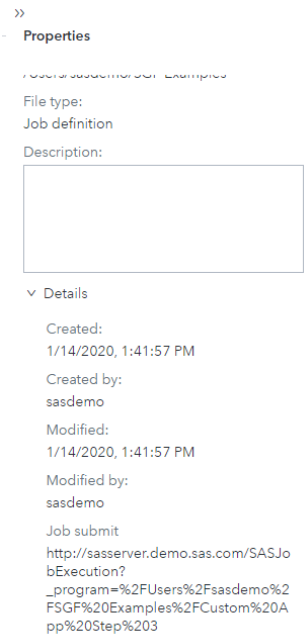


Figure 16. Job Submit Information

Finally, for our dynamic URL, we'll add an %NSTR to mask special characters and mnemonic operators in constant text at macro compilation. Inside parentheses, add the name of the macro variable that we'll send to the next page. The name-value pairs that are added to the end of the URL will be the macro variable name (MakeAndModel) and a concatenation of these two fields that we'll construct in the DATA step code that subsets the SASHELP.CARS table.

```
%let
INFO_URL=?_program=%2FMy%20Folder%2FSGF%20Examples%2F6.%20Complex%20For
m%20Job%20(Part%20Three)&_action=wait%nrstr(&MakeAndModel=);

%Global CarType MaxPrice MinMPG Cyl;

Data work.MyCars (keep = MakeModel MSRP MPG Engine Info) ;
set SASHELP.CARS (rename = (Cylinders = Cylinder Horsepower=Horses));
```

```
length MakeModel Description $100;
length Info varchar(1024);
/* where Type = "&CarType" and MSRP < &MaxPrice & MPG_Highway > &MinMPG
& Cylinder = &Cyl;  Momentarily Disregarding Cylinders */
where Type = "&CarType" and MSRP < &MaxPrice & MPG_Highway > &MinMPG ;
MakeModel = trim(left(Make)) || " " || trim(left(Model));
MPGnum = (MPG_City+MPG_Highway)/2;
MPG= put(MPGNum,$2.);
MSRP = MSRP * 1.20;
Cylinders = put(Cylinder,$1.);
Horsepower = put(Horses,$3.);
Description   =   trim(left(Horsepower))   ||   "   Horsepower   "   ||
trim(left(EngineSize)) || " Liter " || Cylinders || " Cylinder";
Engine = trim(left(Description));
```

We construct our URL variable (Info) by concatenating the server name, the dynamic URL that we built as a macro (INFO_URL), and the actual variable (MakeModel) we just created.

```
Info                                =                                   "<a
href='http://sasserver.demo.sas.com/SASJobExecution/"||"&INFO_URL"||url
encode(strip(MakeModel))||"'>Find This Car</a>";
run;
```

Unlike the prior example, we'll need to place our HTML in PUT statements because we'll need to render the dynamic URL to the dealership listing. For the Kia Rio, the URL will look like this:

http://sasserver.demo.sas.com/SASJobExecution/?_program=%2FUsers%2Fsasdemo%2FSGF%20Examples%2FCustom%20App%20Step%203&MakeAndModel=Kia%20Rio%204dr%20manual

You can add some code to make the output look a bit more polished, but the final step is a PROC PRINT:

```
proc print data=work.mycars;
      var MakeModel MSRP Engine MPG Rating Info;
run;
```

Code Preview

Cars that meet your search criteria

Obs	MakeModel	MSRP	Engine	MPG	Rating	Info
1	Kia Rio 4DR Manual	$12,850	104 Horsepower 1.6 Liter 4 Cylinder	30	★★★	Find This Car
2	Hyundai Accent 2DR Hatch	$13,174	103 Horsepower 1.6 Liter 4 Cylinder	31	★★★	Find This Car
3	Toyota Echo 2DR Manual	$13,450	108 Horsepower 1.5 Liter 4 Cylinder	39	★★★★★	Find This Car
4	Saturn Ion1 4DR	$13,744	140 Horsepower 2.2 Liter 4 Cylinder	31	★★	Find This Car
5	Kia Rio 4DR Auto	$13,944	104 Horsepower 1.6 Liter 4 Cylinder	29	★★	Find This Car
6	Toyota Echo 4DR	$14,113	108 Horsepower 1.5 Liter 4 Cylinder	39	★★★	Find This Car
7	Toyota Echo 2DR Auto	$14,450	108 Horsepower 1.5 Liter 4 Cylinder	36	★★★★	Find This Car
8	Chevrolet Aveo 4DR	$14,613	103 Horsepower 1.6 Liter 4 Cylinder	31	★★	Find This Car
9	Hyundai Accent GL 4DR	$14,799	103 Horsepower 1.6 Liter 4 Cylinder	31	★★★★	Find This Car
10	Hyundai Accent GT 2DR Hatch	$14,924	103 Horsepower 1.6 Liter 4 Cylinder	31	★★★★	Find This Car
11	Suzuki Forenza S 4DR	$15,336	119 Horsepower 2 Liter 4 Cylinder	28	★★★	Find This Car
12	Kia Spectra 4DR	$15,450	124 Horsepower 1.8 Liter 4 Cylinder	28	★★	Find This Car
13	Chevrolet Aveo LS 4DR Hatch	$15,731	103 Horsepower 1.6 Liter 4 Cylinder	31	★★	Find This Car

Figure 17. Car Search Results

STEP THREE – DISPLAY A LIST OF DEALERSHIPS WITH THE CAR AVAILABLE

The dynamic URL created in the prior step allows us to "chain" to another SAS job that will display a fictitious list of local dealerships, with slightly varying prices. Our list should include the name of the car as a title, a stylized table with the MSRP and additional tax and tags, as well as the total cost. To further expand the application, each row in the dealerships table will include a link taking us to a loan calculator. We'll construct that URL just as we did in the prior step, by creating an empty definition for that next step, then rigging the necessary portion from the Display URL property. This will call the loan calculator and pass along the loan amount as a name-value pair that we'll resolve as a macro variable.

```
LOAN_URL=?_program=%2FUsers%2Fsasdemo%2FMy%20Folder%2FSGF%20Examples%2F
6.%20Complex%20Form%20Job%20(Part%20Four)&_action=wait%nrstr(&LoanAMT=)
;
```

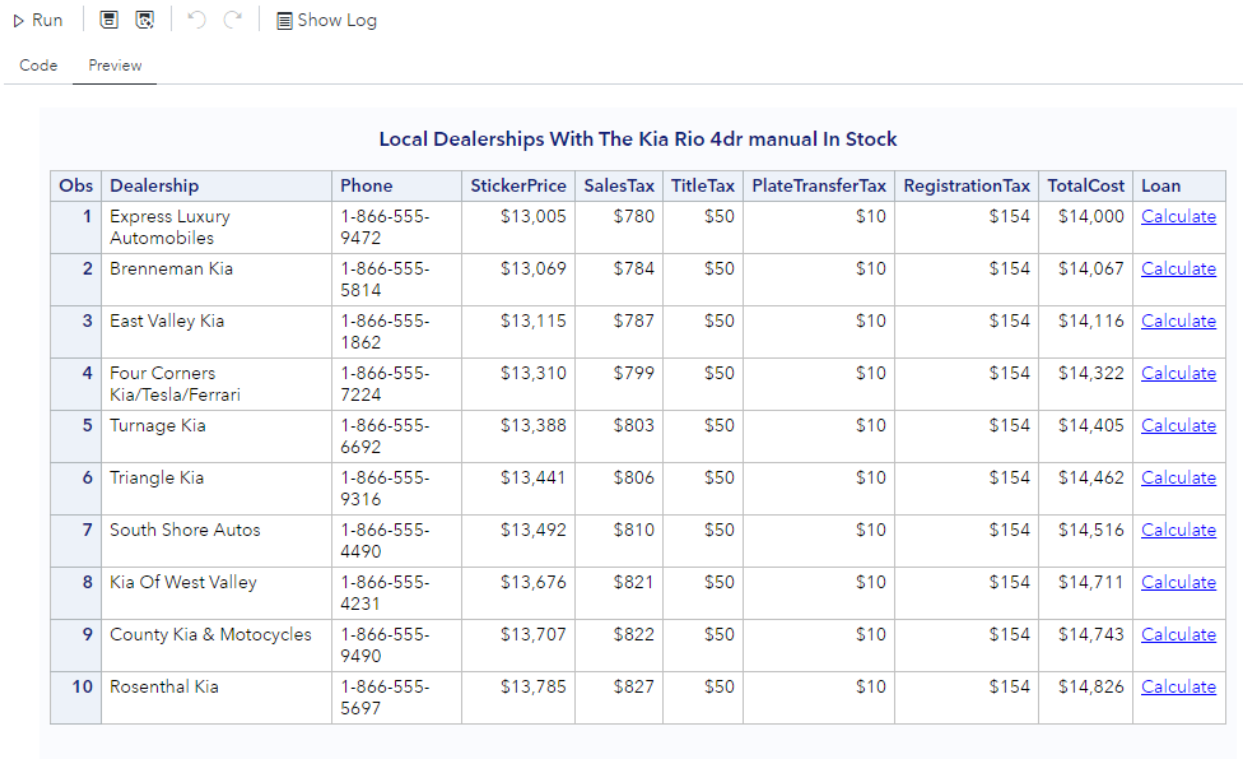
From there, we'll create the dealership table generating random sticker prices so that the loan amounts for each dealership differ slightly.

```
%Global MakeAndModel;

Data work.MyOffers (keep = Make CarURL Loan TotalCost MakeModel
StickerPrice SalesTax TitleTax PlateTransferTax RegistrationTax
Dealership Phone) ;
set work.MyCar ;
      where MakeModel = "&MakeAndModel";
length MakeModel Description CarURL Dealership $100;
length Loan varchar(1024);
MakeModel = trim(left(Make)) || " " || trim(left(Model));
MSRP = MSRP * 1.20; /* Adjusting for inflation */

do i = 1 to 10;
```

```
   x  =  rand("uniform",  1,800);    /*  Creating  Random  upcharge  for
dealerships so prices vary slightly */
   StickerPrice  =  MSRP  +  (x*10);  /*  Creating  Random  upcharge  for
dealerships so prices vary slightly */
   if i = 1 then do;
      Dealership = "Rosenthal " || Make; /* Add Dealership */
   end;
      if i = 2 then do;
      Dealership = "East Valley " || Make;
   end;
      if i = 3 then do;
      Dealership = "Turnage " || Make;
   end;
         if i = 4 then do;
      Dealership = "South Shore Autos";
   end;
         if i = 5 then do;
      Dealership = Trim(left(Make)) || " Of West Valley";
   end;
         if i = 6 then do;
      Dealership = "Triangle " || Make;
   end;
      if i = 7 then do;
      Dealership = "Brenneman " || Make;
   end;
         if i = 9 then do;
      Dealership = "Express Luxury Automobiles";
   end;
         if i = 9 then do;
      Dealership    =    "Four    Corners    "    ||Trim(left(Make))    ||
"/Tesla/Ferrari";
   end;
            if i = 10 then do;
      Dealership = "County " ||Trim(left(Make)) || " & Motocycles";
   end;
    x=  (1111 + floor((1+9999-1111)*rand("uniform"))); /* Create random
last 4 digits for phone number */
         Phone="1-866-555-"||trim(left(x)); /* Create phone number */
      SalesTax = StickerPrice * 0.06;
      TitleTax = 50;
      PlateTransferTax = 10;
      RegistrationTax = (128+180)/2;
      TotalCost   =   SalesTax   +   TitleTax   +   PlateTransferTax   +
RegistrationTax + StickerPrice;
      format   SalesTax   TitleTax   PlateTransferTax   RegistrationTax
StickerPrice TotalCost DOLLAR8.;
      label MakeModel = "Make & Model";
      label StickerPrice = "Sticker Price";
      label Phone = "Phone Number";
      label TotalCost = "Total Cost Of Ownership";

      Loan                              =                              "<a
href='http://sasserver.demo.sas.com/SASJobExecution/"||"&LOAN_URL"||url
encode(strip(TotalCost))||"'>Monthly Payment</a>";
run;

proc print data = work.myoffers;
```

```
var Dealership Phone StickerPrice SalesTax TitleTax PlateTransferTax
RegistrationTax TotalCost Loan;
run;
```

When the link for car is chosen from Step 2, a list like this will be displayed.



Local Dealerships With The Kia Rio 4dr manual In Stock

Obs	Dealership	Phone	StickerPrice	SalesTax	TitleTax	PlateTransferTax	RegistrationTax	TotalCost	Loan
1	Express Luxury Automobiles	1-866-555-9472	$13,005	$780	$50	$10	$154	$14,000	Calculate
2	Brenneman Kia	1-866-555-5814	$13,069	$784	$50	$10	$154	$14,067	Calculate
3	East Valley Kia	1-866-555-1862	$13,115	$787	$50	$10	$154	$14,116	Calculate
4	Four Corners Kia/Tesla/Ferrari	1-866-555-7224	$13,310	$799	$50	$10	$154	$14,322	Calculate
5	Turnage Kia	1-866-555-6692	$13,388	$803	$50	$10	$154	$14,405	Calculate
6	Triangle Kia	1-866-555-9316	$13,441	$806	$50	$10	$154	$14,462	Calculate
7	South Shore Autos	1-866-555-4490	$13,492	$810	$50	$10	$154	$14,516	Calculate
8	Kia Of West Valley	1-866-555-4231	$13,676	$821	$50	$10	$154	$14,711	Calculate
9	County Kia & Motocycles	1-866-555-9490	$13,707	$822	$50	$10	$154	$14,743	Calculate
10	Rosenthal Kia	1-866-555-5697	$13,785	$827	$50	$10	$154	$14,826	Calculate

Figure 18. Dealership Listing

ADDITIONAL STEPS

While a list of cars is great, it might help to add functionality to calculate the estimated monthly payment with a loan calculator. In this scenario, the dynamic hyperlink to the dealership table that sends the total cost to another SAS job enables users to select loan terms (years, APR) and calls a final job that displays a breakdown of the monthly cost.

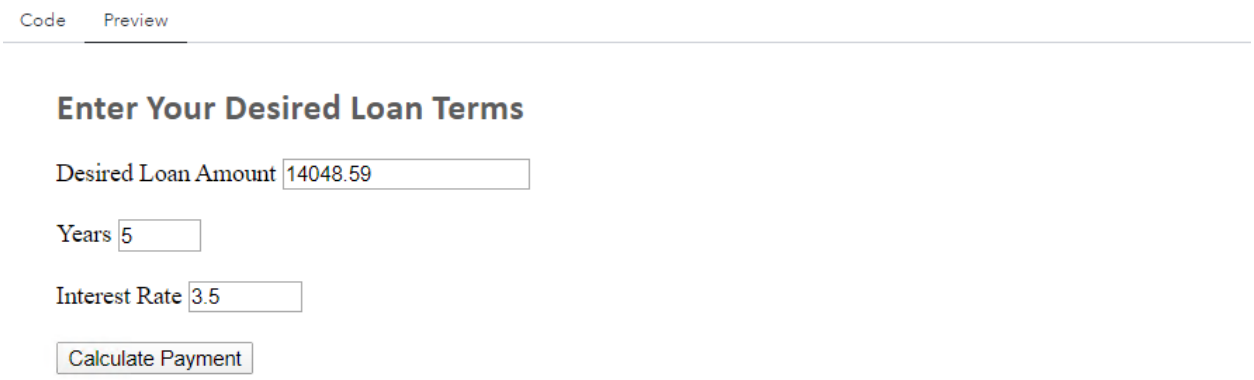


Figure 19. Loan Calculator Parameter Selection Screen

The loan amount is passed over as a macro variable and added to a form that enables users to select years to finance the car and an interest rate. This is then sent to the final job to display the loan information.

Figure 20. Loan Information

CONCLUSION

With SAS Viya jobs, you can marry your wealth of SAS knowledge with a bit of HTML and create dynamic web-based applications.

REFERENCES

Full Code Referenced In This Paper On SGF2020 GitHub

https://github.com/sascommunities/sas-global-forum-2020/tree/master/papers/4185-2020-Anderson

SAS Institute Inc. 2019. SAS Studio 5.2: User's Guide. Cary, NC: SAS Institute Inc.

https://documentation.sas.com/?docsetId=webeditorug&docsetTarget=titlepage.htm&docsetVersion=5.2&locale=en

SAS Institute Inc. 2019. SAS Job Execution Web Application 2.1 User's Guide. Cary, NC: SAS Institute Inc. https://documentation.sas.com/?docsetId=jobexecug&docsetTarget=titlepage.htm&docsetVersion=2.2&locale=en

CONTACT INFORMATION

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

Robert L. Anderson II
SAS Institute Inc.
Robert.Anderson@sas.com
LinkedIn: robert-l-anderson-ii