PharmaSUG 2024 - Paper SD-179

Developing Web Apps in SAS Visual Analytics

Jim Box and Samiul Haque, SAS institute

ABSTRACT

SAS® Viya ® provides capabilities to develop web applications that let you use HTML pages to provide inputs to programs that use SAS, R, and/or Python to create, display, and share analysis results across your organization. The GUI uses standard HTML programming to collect the inputs and the system leverages the existing scalability and security of your cloud environment.

INTRODUCTION

The SAS Job Execution Web Application is a way to deliver results of SAS code in an interactive manner based on inputs captured from the end user. It's very similar to stored processes in capabilities, and it has the same sort of interactive capabilities of some R Shiny ® applications. To make these work, you'll need some SAS code (which can contain R and/or Python Code) and a little bit of HTML code for the interface (which ChatGPT can be useful for writing).

EXAMPLE – SIMPLE RANDOMIZATION LIST CREATION

It's easier to see what we are doing if we start with an example. We wanted to make a job that would prompt the user for some specifications for a simple randomization list, run some code to create the list, then send the output tables to a Visual Analytics ® page that would have some imbedded visualizations to do a QC of the process. As a bonus, we made one of those visualizations in R.

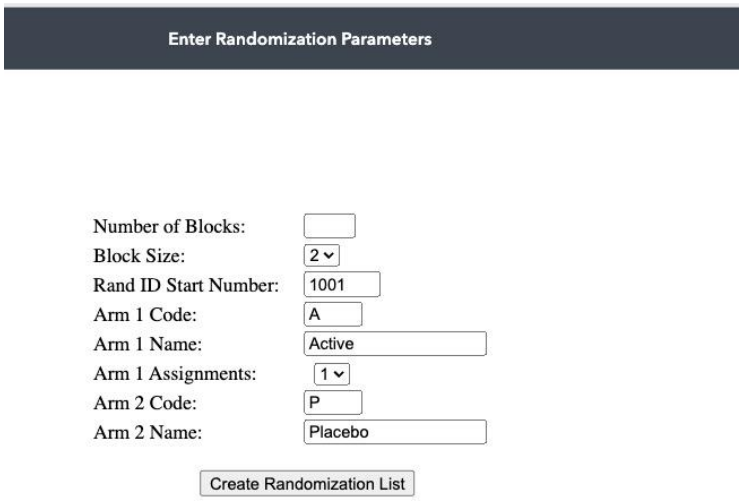


Figure 1: Parameter Collection

The first part of the application is the parameter collection screen (Figure 1). This is a simple HTML file. Each one of these values collected on the form will be passed to the SAS Job in the form of macro variables that will be used in the SAS program that produces the results.

Figure 2: VA Report of output

Figure 2 shows the output report. Clockwise from the upper left there's a table of the parameters captured from the input form, a bar chart showing the randomization breakdown by block, a bar chart with the overall randomization counts, and then finally the actual randomization list. Let's look under the hood to see how we came up with this.

BUILDING THE JOB

DEVELOP THE SAS CODE

The first thing to do is to start with your actual SAS program and figure out what inputs you need to execute. Figure 1 shows the input parameters we will need to create the randomization scheme.

Here's the code we used to capture the parameters:

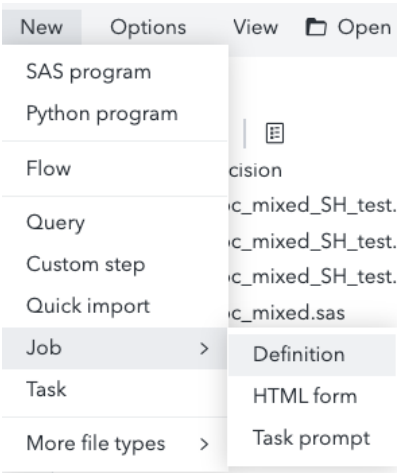
```
%let Blocks = &_blocks;
%let size = &_size;
%let t1 = &_t1;
%let t2 = &_t2;
%let s1 = &_s1;
%let s2 = %eval(&size - &s1);
%let n1 = &_n1;
%let n2 = &_n2;
%let rid = &_rid;

%let st=%sysevalf(&S1 +&S2);
%let C1 = %sysevalf((&S1*&size/&St));
```

Figure 3: Capturing parameters

To make it easier to differentiate between parameters passed by the UI and the macro variables, we used a leading underscore when setting up the parameter capture. Those values already come in as SAS macro variables with the parameter names, so we didn't have to do this step, but it makes it easier to track things of you do. It was also helpful to test the code by putting values in for the parameters.

The next section of code is just producing the randomization lists in the regular manner. Once the randomization is done, we make two output datasets, one called RandList with the output of the randomization and one called RandParm, which captures the values of the parameter.

CONFIGURE THE JOB



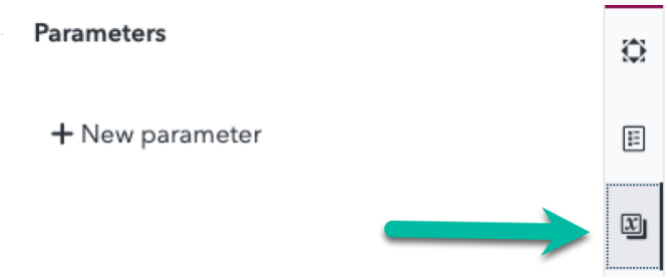


Figure 4: Create a new job in SAS Studio **Figure 5:** Setting Parameters

To create a new job, in SAS studio select a Job Definition from the New menu (Figure 4). This will give you the place to enter in your program. On the right-hand menu, you will see the Job Parameters icon (Figure 4), which will open the parameters definitions. You will need to define some standard parameters every time you run a job: an action parameter and an output type parameter. Figure 6 shows how we have set them up.

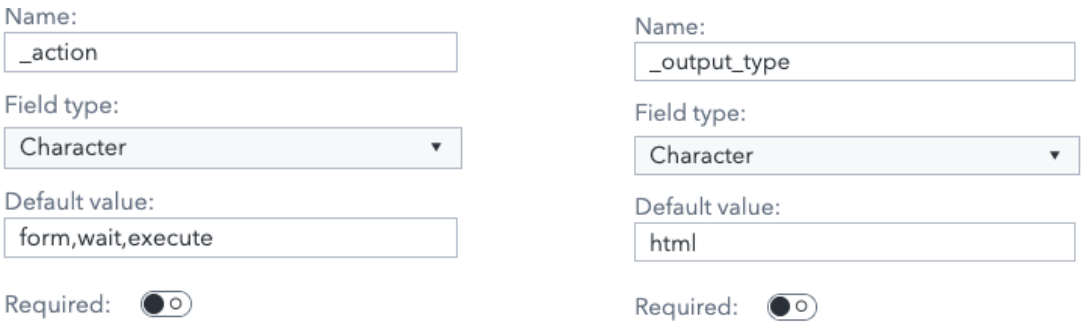


Figure 6: Required Parameters

These parameters are set up to use a html form to capture the inputs for this job. We also set up an additional parameter called _contextName so we can use a compute context that is always running, so the job does not need to spin up a new SAS compute context every time.

From this point, we need to define a new parameter for everything we want in our form (figure 1). Finally, we need to define the url for the job, and generate some html to show when the job runs.

The url definition will look something like this:

```
%let url= %nrstr(/SASJobExecution/?_program=/Public/RandomizationGeneration);
%let reload_url="&url.";
```

For the html, used a data step to push the html to the form window (Figure 7):

```
138  /* Send code that the project is done */
139
140  data _null_;
141  file _webout;
142  put '<!DOCTYPE html><html lang="en"> <head><title>Two-Arm Block Randomization</title><style type="text/css">
143  @font-face {
144    font-family:AvenirNext;
145    src:url("/SASJobExecution/images/AvenirNextforSAS.woff") format("woff");
146  } body, input, select {
147    font-family: AvenirNext,Helvetica,Arial,sans-serif;
148    text-rendering: optimizeLegibility;
149    -webkit-tap-highlight-color: rgba(0,0,0,0);
150  } .pointer {
151    cursor: pointer;
152  } u {
153      font-size: 15pt;
154  }</style></head> <body role="main"> <h1><center>';
155  put '
156  Randomization List Generated</h1>
157  ';
158  put ' <script>
159      setTimeout(function(){
160         window.location.href =
161  ';
162  put "%bquote(&reload_url)";
163  put ';
164      }, 2500);</script> </html> ';
165  run;
```

Figure 7: HTML code returned after job executes

All this to put the message “Randomization List Generated” once the code runs (line 156).

CREATE THE INPUT FORM

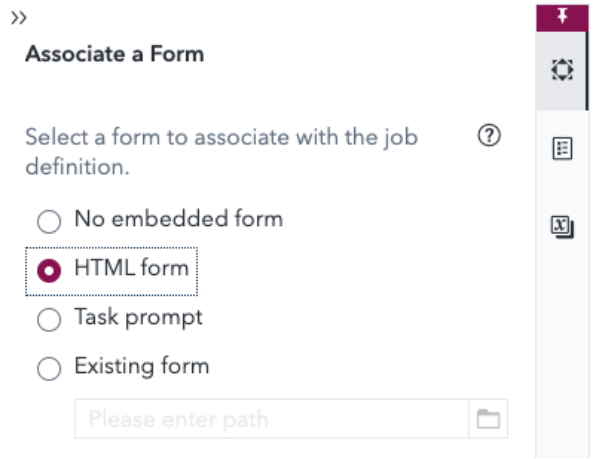


Figure 8: Associating a Form

Next, we need to associate an html form (Figure 8). We'll create a new one in the job definition itself; you can see the other options available. To write the html, we called on ChatGPT to help; it's a pretty basic form. Figure 9 shows the first section.

Program HTML form Preview

```
1
2   <!DOCTYPE html>
3   <html lang="en">
4
5    <body role="main">
6   <center> <!--h4>Enter Randomization Parameters</h1--> <form id="entry_form" action="/SASJobExecution/">
7   <input type="hidden" name="_program"     value="$PROGRAM$"/>
8   <input type="hidden" name="_action"      value="wait,execute"/> <br/>
9   <table> <tr>
10      <td>Number of Blocks: </td>
11      <td>
12      <input type="number" name="_blocks" min="1" max="90" step="1" required>
13      </td>
14  </tr>
15  <tr>
16      <td>Block Size: </td>
17      <td>
18      <select name="_size">
19        <option value="2">2</option>
20        <option value="3">3</option>
21        <option value="4">4</option>
22        <option value="5">5</option>
23        <option value="6">6</option>
24        <option value="7">7</option>
25        <option value="8">8</option>
26      </select>
27      </td>
28  </tr>
```

Figure 9: Input form HTML

Line 12 shows us capturing the number of blocks into the _blocks parameter that we defined similar to Figure 5, and line 18 is capturing the block size parameter _size from a pull-down selection as seen in Figure 1.

To test your html, Run the job, and a Preview pane will appear where you can see the output (figure 10).

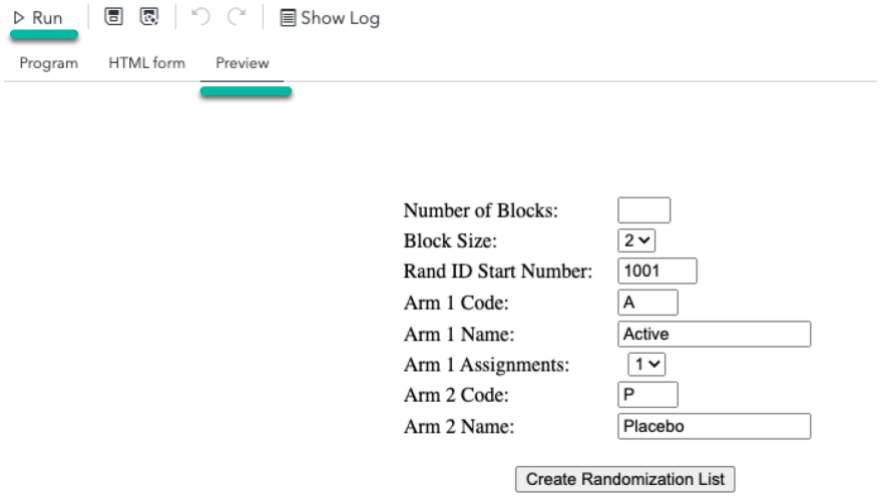


Figure 10: HTML preview

CREATE THE R OUTPUT

Creating the R output (figure 2, bottom right) wasn't particularly difficult, as you can use ggplot in PROC IML to make the graphic (Figure 11).

```
166
167  /* Create an R barplot for showing overall breakdown  */
168  proc iml;
169  /* Convert SAS Table to R data frame */
170  call ExportDataSetToR("RandList", "df" );
171
172  /* Submit R Code */
173  submit  / R;
174  library(ggplot2)
175
176  png(file="/data/compute-landingzone/www/images/RBarRand.png",
177  width=600, height=350)
178
179  ggplot(df, aes(x=Tx)) +
180    geom_bar(aes(fill = factor(Txn))) +
181    geom_text(aes(label =after_stat(count)), stat = "count", vjust = 1.5, color = "black") +
182    theme(legend.title=element_blank())
183
184  endsubmit;
185
186  quit;
```

Figure 11: PROC IML code to make a graphic

What's important here is where we saved the png output file. Line 176 shows where on the Viya server we saved the output. Our Viya administrator set up a simple web server on the machine, this was important to be able to display the graphic on the VA report. Because VA cannot automatically update an imbedded graphic, to get this image to update on the report, we had to call it from an html file that would continuously update (Figure 12). This html file (line 191) can be imbedded in the VA report.

```
190  data _null_;
191  file "/data/compute-landingzone/www/images/RBarRand.html";
192
193  put'<body> ';
194  put'    <img id="RBarRand" src="/images/RBarRand.png"> ';
195  put /;
196  put'    <script> ';
197  put'        // Get a reference to the image element ';
198  put'        const img = document.getElementById("RBarRand"); ';
199  put /;
200  put'        // Define a function to refresh the image ';
201  put'        function refreshImage() { ';
202  put'            img.src = "/images/RBarRand.png?" + new Date().getTime(); ';
203  put'        } ';
204  put /;
205  put'        // Call the refreshImage function every 5 seconds ';
206  put'        setInterval(refreshImage, 5000); ';
207  put'    </script> ';
208  put /;
209  put'</body> ';
210
211  run;
```

Figure 12: HTML File that continuously updates

CREATE THE VA REPORT

We have everything we need for the VA report now:

1. HTML input form
2. Parameter Dataset
3. Randomization List dataset
4. HTML form linking to the R graphic

The Randomization List Parameters and Randomization List objects (Figure 2) are just basic VA list tables – the only thing specific to this report is that in the options pane, we used the Periodically reload object data selection and set it to 5 seconds (Figure 13). We did the same thing for the Treatments By Block Assignment bar chart.

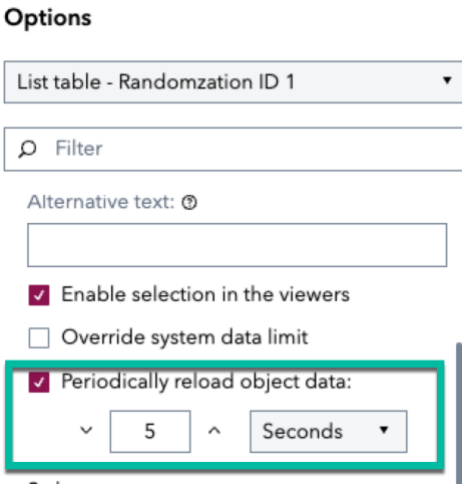


Figure 13: Setting object to automatically reload

The R bar chart is just a Web Content object, and all we had to do was point the URL to where we wrote out the html file (Figure 12, line 191). This is where we needed help from the admin who set up the web server on our Viya instance. We needed the location and a port number. The URL itself wound up looking like this:

```
https://viya4.globalhls.sashq-d.openstack.sas.com:8443/images/RBarRand.html
```

Everything to the left of the colon is just the URL for our Viya instance; everything to the right is the port number for the web service and the location we wrote the html file out to.

FINALIZING THE APP

Now there are two things left to do: setting up a page to call the application and linking to it from the VA report. First, we need to make a new report tab, and set it to Pop-Up8 (Figure 14)

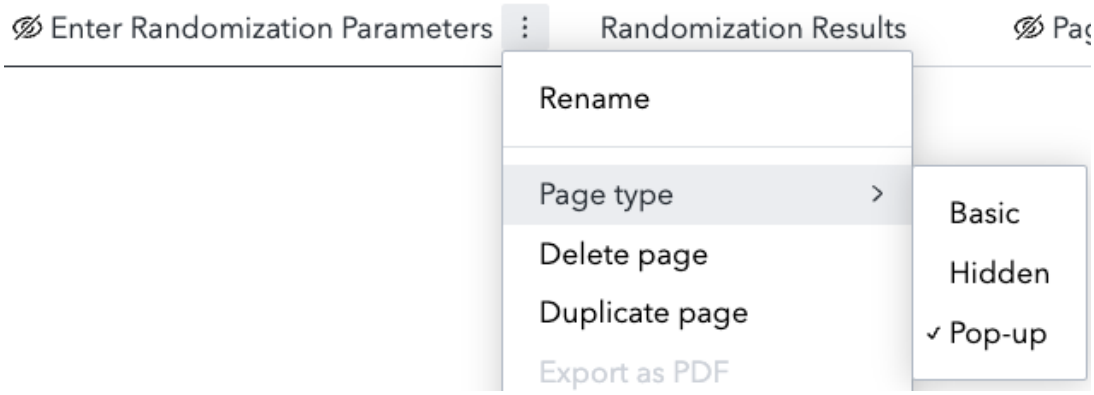


Figure 14: Creating a new Pop-up Page

This page will then just have a web content object on it, with the URL we defined earlier and use that in the URL field:

```
https://viya4.globalhls.sashq-d.openstack.sas.com/SASJobExecution/
?_program=%2FPublic%2FRandomizationGeneration
```

The last thing to do is to go onto our main VA report page and enter a text object, and go to Links .. Add page link (Figure 15):

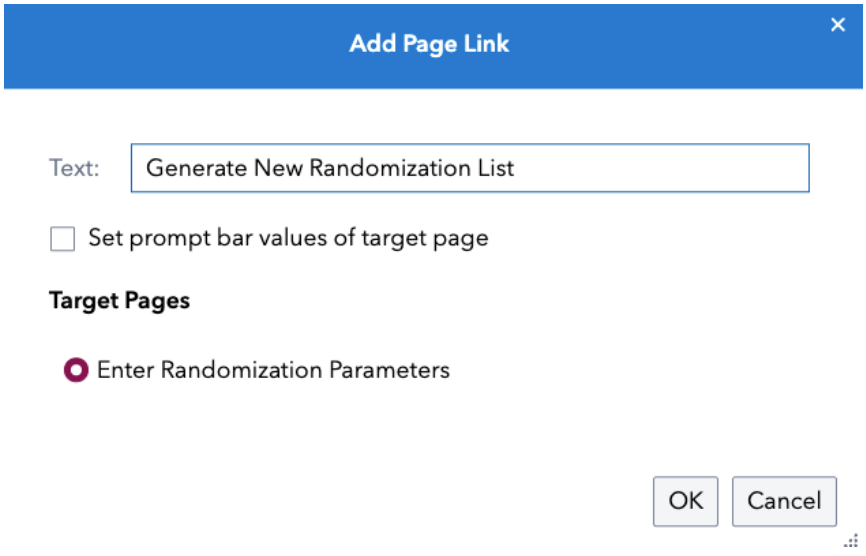


Figure 15: Adding a Page Link

This will add the hyperlink we see on the very top of figure 2 and will launch the parameter collection form (figure 1).

CONCLUSION

The SAS Jobs Execution Web Application is an impressive way to create and share applications built on SAS, R, and Python code. With code and HTML, you can build our interactive applications for your users.

ACKNOWLEDGMENTS

The authors would like to thank Wouter Van de Weghe, the SAS admin we would not be able to do anything without.

RECOMMENDED READING

- SAS Documentation: https://go.documentation.sas.com/doc/en/pgmsascdc/default/jobexecug/titlepage.htm

CONTACT INFORMATION

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

Jim Box	Samiul Haque
Jim.box@sas.com	Samiul.Haque@sas.com
https://www.linkedin.com/in/jwbox/	https://www.linkedin.com/in/samiulhaque/