SAS4309-2020

SAS® Visual Analytics SDK: Embed SAS Visual Analytics Insights in Your Web Pages and Web Apps

Brad Morris, Robby Powell, SAS Institute Inc.

ABSTRACT

Embedding SAS® Visual Analytics insights in your web pages and web apps lets you share insights through the portals your users regularly access. You already use SAS Visual Analytics to gain insights through drag-and-drop interactions and rich visualizations of your analytics results. And you share those insights with others to collaborate and solve challenges. The next step is embedding your insights in your corporate portal to increase the number of eyes that see what you see. Welcome to the SAS® Visual Analytics SDK! The SAS Visual Analytics SDK is a collection of JavaScript libraries that web developers can use to embed SAS Visual Analytics insights within their web pages and web apps. Add to your web pages and web apps entire SAS Visual Analytics reports or individual objects from your reports. Join this session to learn about the SAS Visual Analytics SDK and see live demos of how easy it is to embed live, interactive SAS Visual Analytics content in your web pages and web apps.

INTRODUCTION

This paper presents an overview of the SAS® Visual Analytics SDK and provides the details you need to begin publishing your SAS® Visual Analytics insights in your custom web pages, web apps, and portals. First you will learn the configuration changes needed for SAS® Viya®. Just minor tweaks, but these modifications are required to allow the SAS Visual Analytics SDK to access your report content.

Next you receive step-by-step instructions guiding you through the SAS Visual Analytics no-code approach to embedding content on the web. Using the declarative HTML custom element tags: <sas-report>, <sas-report-page>, <sas-report-object>, you will understand the basics and immediately be able to put them to use.

The advanced topics section introduces using the JavaScript API to add SAS Visual Analytics insights to create dynamic web experiences. Also included in this section is an explanation of how to use the SAS Visual Analytics data-driven content object to dynamically build interactions between your SAS Visual Analytics SDK objects and third-party data visualizations – within your custom portal, and how to pass parameters from your custom web app to your SAS Visual Analytics reports.

After reading this paper, you will possess all the tools you need to begin embedding SAS Visual Analytics content and insights into your custom web pages and web apps.

GETTING STARTED WITH THE SDK

The SAS Visual Analytics SDK, here forth referred to as “SDK”, allows you to embed your SAS Visual Analytics content on the web, as either a full report or its individual parts. The SDK achieves this by connecting to a SAS Viya server to access the report content and data, and then renders interactive report content directly in your HTML page. Accessing a SAS Viya server from the SDK requires some basic server configuration, but after this is completed you can begin embedding your SAS Visual Analytics insights with just a few lines of HTML.

SAS VIYA SETUP

Guest Access

The SDK optionally allows connection to the SAS Viya server using guest access, bypassing the need for a user to explicitly authenticate. This requires that the SAS Viya system be set up to enable guest access, and that the guest user has access to the appropriate report content and data. For more information on setting up guest access, see SAS Help Center.

Cross-Origin Resource Sharing (CORS)

It is likely that your web page will reside on a domain that is different from the domain of the SAS Viya server, which means that the SDK will be making cross-origin requests. The default SAS Viya configuration disallows cross-origin resource sharing (CORS) and must be explicitly enabled to allow communication with the SDK. CORS is controlled by a whitelist of allowedOrigins, so it is necessary to add your site's origin to this configuration property. See developer.sas.com and SAS Help Center for more explanation of CORS detailed steps to update the configuration.

Cross-Site Request Forgery (CSRF)

SAS Viya servers protect against Cross-Site Request Forgery (CSRF) by blocking requests where the HTTP Referer Header does not match the URI of the requested resource or a URI whitelist. Similar to CORS, the origin of the site using the SAS Visual Analytics SDK must be whitelisted in the CSRF configuration. See SAS Help Center for more information. CSRF checks are only made on authenticated sessions, so this is not applicable if using guest access.

ANATOMY OF THE HTML

The SAS Visual Analytics SDK is a JavaScript library, but that doesn't mean that you actually need to know JavaScript to use it. In fact, through the use of our custom web components, you can embed a SAS Visual Analytics report into a web page without writing a single line of JavaScript!

Loading the SDK library

First, your HTML page needs to load the va-report-components script, which gives it access the SDK APIs and custom HTML elements that are defined by the library. The library is published to Node Package Manager as @sassoftware/va-report-components and can then be accessed by any content delivery network (CDN) that mirror's NPM content. This example SCRIPT tag shows how to include the 0.3.1 version of the library from the unpkg.com CDN:

```
<script async src="https://unpkg.com/@sassoftware/va-report-
components@0.3.1/dist/umd/va-report-components.js"
></script>
```

The version number specified follows semantic versioning, so you can specify @latest to get the latest version, or @0.3 to get the latest 0.3.* patch version. This can be convenient when trying out the SDK in a development environment, but it is advised to specify the full *major.minor.patch* version when deploying production code.

Adding report content

There are three custom HTML elements that are defined by the SDK: sas-report, sas-report-page, and sas-report-object. As their names suggest, they each give you the ability to target different levels of report content for inclusion in your page. These elements are all fully interactive, allowing you to recreate the SAS Visual Analytics report experience wherever it is needed.

sas-report

The sas-report element is used to insert an entire report into the body of an HTML page. sas-report requires only two attributes; *url* and *reportUri*. *url* is the URL of the SAS Viya server and should contain the protocol, hostname, and optionally the port. *reportUri* is the uniform resource identifier of the report that you are embedding. This information can be accessed most easily through the "Copy link…" option in SAS Visual Analytics, which is covered later in this paper. A sas-report element looks like this:

```
<sas-report
  url="https://my-viya-server.com"
  reportUri="/reports/reports/c3c6befb-3981-4c9e-b011-7dc11dec5e37"
</sas-report>
```

sas-report-page

The sas-report-page element is similar to sas-report except that it embeds a single report page instead of the entire report. It requires *url* and *reportUri* attributes, but additionally requires either *pageIndex* or *pageName* in order to select the page to be displayed.

pageIndex is a zero-based index specifying which page to display. This parameter is the easiest to specify, since you don't have to know the unique page identifier, but if a report is edited and a page's index is changed *pageIndex* will now reference the new *nth* page. *pageName* is the unique page identifier, which can easily be found using the SAS Visual Analytics "Copy link…" option.

If a report page includes controls in the page prompt area, those will be shown as part of the sas-report-page element. If a report includes controls in the report prompt area, those will not be shown when using sas-report-page. The page will be displayed using the report prompt control's default values. A sas-report-page element referencing the third page of a report looks like this:

```
<sas-report-page
  url="https://my-viya-server.com"
  reportUri="/reports/reports/c3c6befb-3981-4c9e-b011-7dc11dec5e37"
  pageIndex="2"
</sas-report-page>
```

sas-report-object

The sas-report-object element enables you to embed a single report object into your HTML page. The object can be either a single visual or a container of visuals.

sas-report-object requires the *url* and *reportURI* attributes, as well as an additional *objectName* attribute. *objectName* is the unique object identifier for the visual or container that you are referencing in the report. It can be accessed using the "Copy link…" option in SAS Visual Analytics.

The sas-report-object retains the interactivity of the objects, as defined in the report, even across objects. For example, a filter or linked selection action defined between two report objects will be honored if both the source and target objects are added to the HTML page using the sas-report-object. A sas-report-object element looks like this:

```
<sas-report-object
  url="https://my-viya-server.com"
  reportUri="/reports/reports/c3c6befb-3981-4c9e-b011-7dc11dec5e37"
```

```
    objectName="ve27"
  </sas-report-object>
```

SDK HTML TAG GENERATION

While you could write SDK element tags by hand, it is often easier to use the SAS Visual Analytics UI to automatically generate them for you. This can be done using the Copy Link feature, which is available from the SAS Visual Analytics report menu as well as the page and object menus.

Open the report that you wish to embed, and then click "Copy Link…" from the report menu.

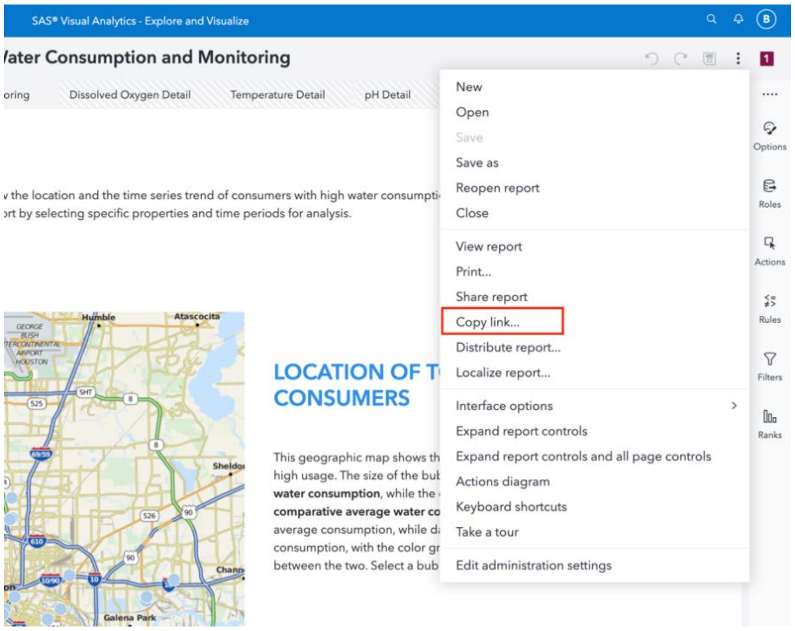
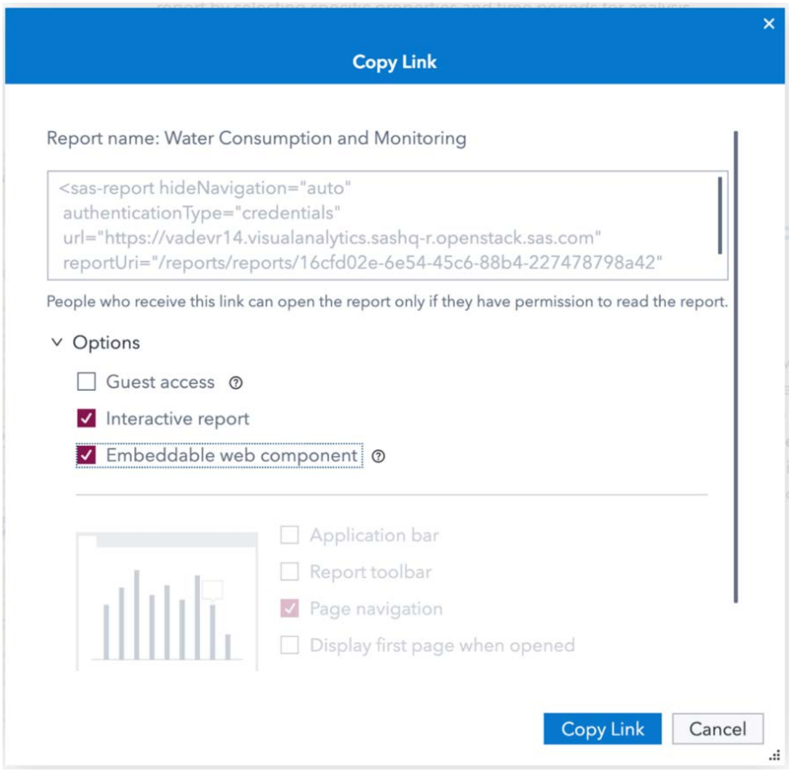


Figure 1: The "Copy Link" Report Menu Option

This displays the Copy Link window, which generates a sas-report HTML element tag when you select the "Embeddable web component" option.


Copy Link

Report name: Water Consumption and Monitoring

```
<sas-report hideNavigation="auto"
authenticationType="credentials"
url="https://vadevr14.visualanalytics.sashq-r.openstack.sas.com"
reportUri="/reports/reports/16cfd02e-6e54-45c6-88b4-227478798a42"
```

People who receive this link can open the report only if they have permission to read the report.

Options

- [ ] Guest access
- [x] Interactive report
- [x] Embeddable web component
- [ ] Application bar
- [ ] Report toolbar
- [x] Page navigation
- [ ] Display first page when opened

Figure 2: The Copy Link Window

The *url* and *reportUri* arguments are populated for you, and the HTML tag can then be pasted directly into your page. The same can be done for pages, containers, and objects by using “Copy Link...” menu item from each respective menu. The page menu can be accessed from the report page tab, and the object and container menus can be accessed from their object toolbar.

SDK STYLING

CUSTOM ELEMENT STYLES

The sas-report, sas-report-page, and sas-report-object custom elements all extend from the built-in HTMLElement and therefore support all of the same CSS properties for setting element styling. The SDK elements have no default width and height, so it is up to you use CSS or inline styles to set the size according to your needs. That could be as simple as setting a hard-coded pixel size, or as complex as using Flex box or CSS grid layout to affect the size of the element. Here is a simple example using inline style:

```
<sas-report
  url=”https://my-viya-server.com”
  reportUri="/reports/reports/c3c6befb-3981-4c9e-b011-7dc11dec5e37"
  style=”width: 100%; height: 400px”
</sas-report>
```

The sas-report also supports a few custom style properties that allow you to change the border and background color of the page tabs, allowing you to make the embedded report better match the styling of your page. These style properties are *--sas-report-padding* and --*-sas-report-background-color*. Here is the syntax showing these custom style properties:

```
<sas-report
  url=”https://my-viya-server.com”
```

```
    reportUri="/reports/reports/c3c6befb-3981-4c9e-b011-7dc11dec5e37"
    style=”--sas-report-padding:16px;--sas-report-background-color:#0378cd;”
</sas-report>
```

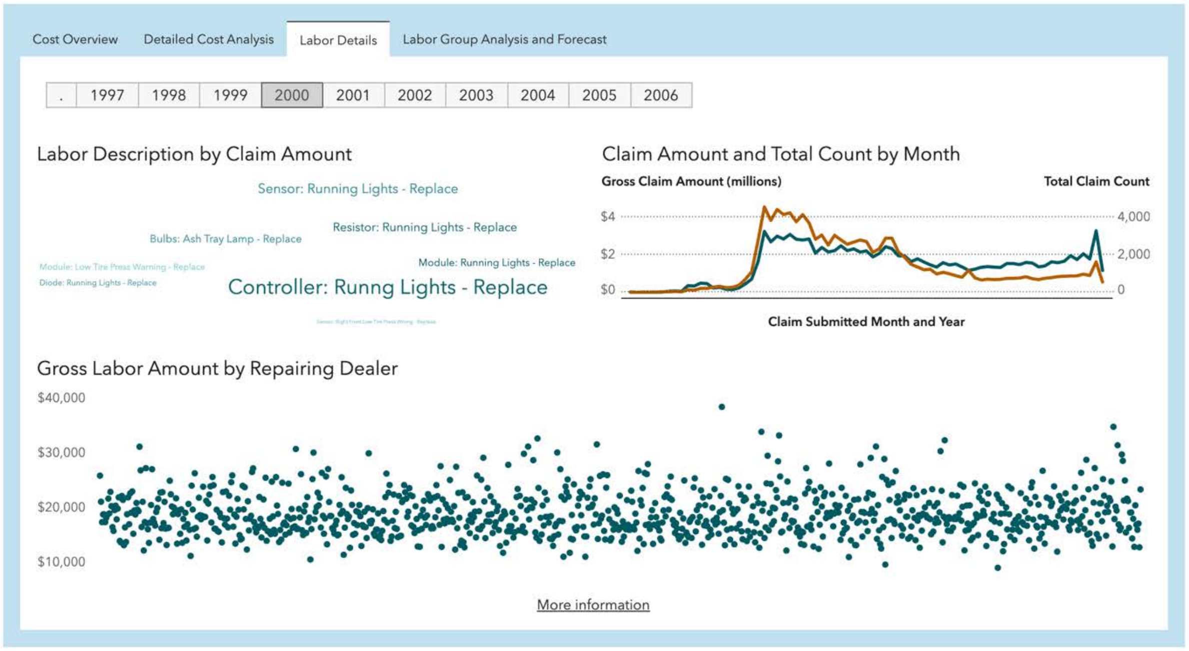
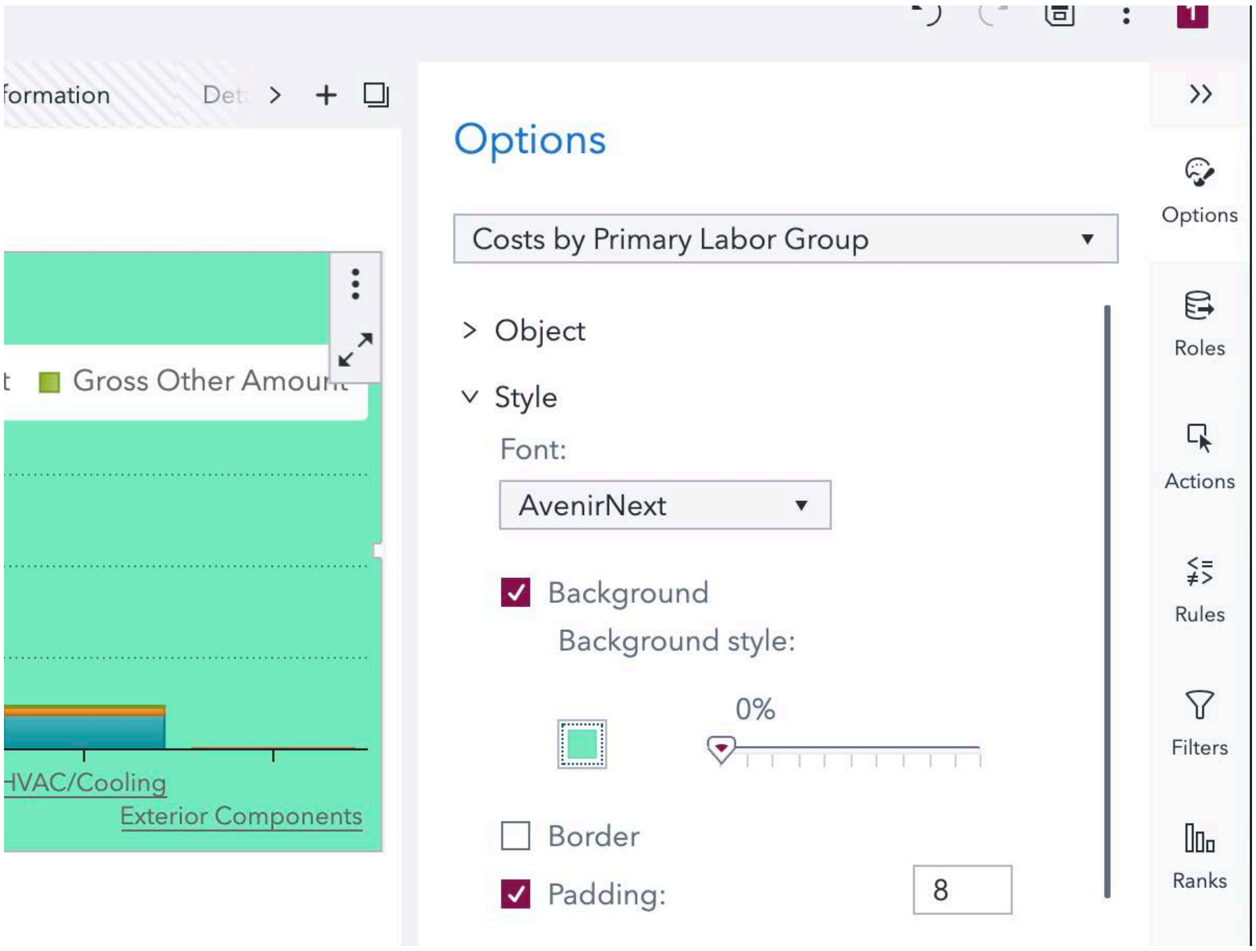

Cost Overview
Detailed Cost Analysis
Labor Details
Labor Group Analysis and Forecast
. 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006
Labor Description by Claim Amount
Sensor: Running Lights - Replace
Resistor: Running Lights - Replace
Bulbs: Ash Tray Lamp - Replace
Module: Low Tire Press Warning - Replace
Module: Running Lights - Replace
Diode: Running Lights - Replace
Controller: Runng Lights - Replace
Claim Amount and Total Count by Month
Gross Claim Amount (millions)
Total Claim Count
$4
$2
$0
4,000
2,000
0
Claim Submitted Month and Year
Gross Labor Amount by Repairing Dealer
$40,000
$30,000
$20,000
$10,000
More information

Figure 3: Custom Style Properties on the sas-report Element

CUSTOM REPORT STYLES

Styling of elements inside of a report are entirely controlled by the report style options that are applied to the report when created in SAS Visual Analytics. Style customizations can be made by changing the report theme or by setting specific style properties using the *style* options in the *Options* side panel in SAS Visual Analytics.


Options

Costs by Primary Labor Group

Object

Style

Font: AvenirNext

✓ Background

Background style: 0%

Border

✓ Padding: 8

Figure 4: Setting Report Style Properties

AUTHENTICATION

The SDK authenticates to SAS Viya using the SAS Viya Logon service. The SDK provides the optional attribute *authenticationType*, which supports two types of authentication, *guest* and *credentials*, which controls how the SDK authenticates the user with the SAS Viya server. The authenticationType attribute can be set on any of the SDK custom elements. This attribute is not required, and defaults to *credentials*.

GUEST

authenticationType=guest automatically authenticates using the Guest user. As described in the SAS Viya Setup section, SAS Viya Guest Access must be enabled on the server, which includes setting up all authorization rules for the guest user. It is important to make sure that the source report and its data are both accessible to the guest user. Guest authentication can be used in scenarios where the SDK is accessed through a public facing page, or where the viewers will not have individual access to the SAS Viya system. When using guest authentication, there is no intervention necessary by the SDK report consumer and the authentication process will be initiated and completed automatically by the SDK.

CREDENTIALS

authenticationType=credentials authenticates with the SAS Viya server using the account credentials supplied by the user who is visiting the page that is embedding the report. This method of authentication utilizes the SAS Viya Logon service to establish session authentication, storing the session cookies in the browser for future page visits. This is the same authentication flow that happens when you visit the SAS Visual Analytics application directly. Using this authentication mode, the first page visit will result in the visitor needing to authenticate with the SAS Viya server. The SDK will detect that the user is not authenticated and will prompt the user to log in. Clicking the login button will then pop up a window allowing the user to log in directly to the SAS Viya server.

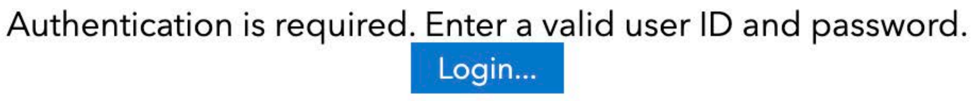

Authentication is required. Enter a valid user ID and password.

Figure 5: Login Prompt

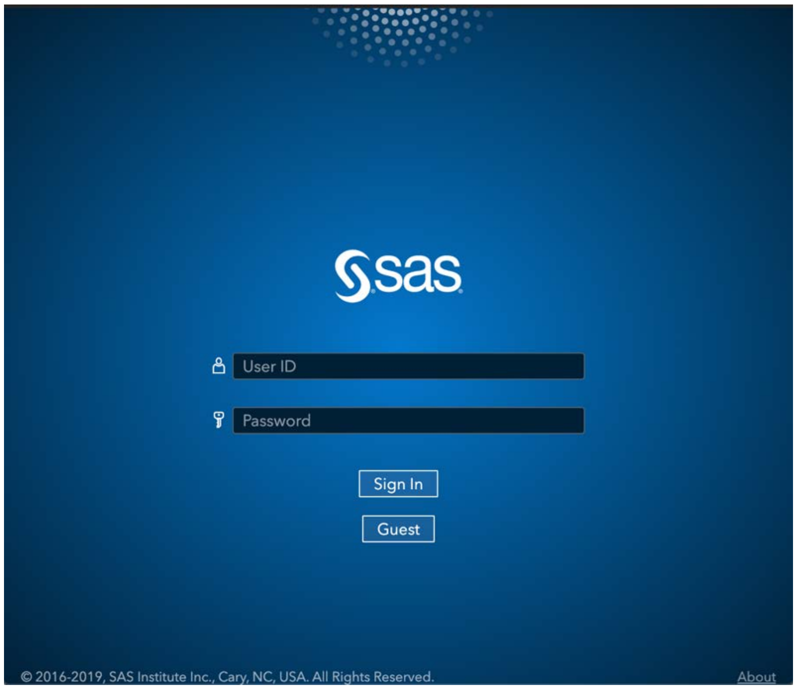

sas
User ID
Password
Sign In
Guest
© 2016-2019, SAS Institute Inc., Cary, NC, USA. All Rights Reserved.
About

Figure 6: SAS Logon Manager

Once the user has entered their credentials and successfully logged in, the SDK will automatically resume connection with the SAS Viya server and load the sas-report, sas-report-page, and sas-report-object elements that are visible on the page. As long as the authenticated session has not expired reloads of the page will automatically authenticate by using the session cookies stored in the browser. If you have configured SAS Logon Manager for single-sign-on, then this will just work with the SDK without any additional configuration changes.

JAVASCRIPT API

The API that has been introduced to this point is the declarative HTML custom element tag API. This API is a great way to start developing with the SDK, as it allows you to easily understand the structure of report content and how it fits into the page DOM. This works fine for pages with static page content, but if you want to create a more dynamic web experience you will likely start to incorporate JavaScript that modifies the DOM structure and changes the content on your page. Doing this with report content requires using the SDK's JavaScript API.

GLOBAL OBJECT AND THE LOAD EVENT

The SDK API is accessible through the global *vaReportComponents* object. This object is defined by the SDK library, but it is not available for use until the library has finished loading. You can get notified of when the SDK is ready to use by listening to the *vaReportComponent.loaded* global event:

```
<script>
  window.addEventListener('vaReportComponents.loaded', function() {
    // The SAS Visual Analytics SDK is loaded and ready
   // Do any necessary SDK related page initialization here.
  });
</script>
```

Once the load event has been fired you can access the SDK JavaScript API, which includes the ability to create sas-report, sas-report-page, and sas-report-object elements through JavaScript. These elements are available on the *vaReportComponents* object as *SASReportElement*, *SASReportPageElement*, and *SASReportObjectElement*. Here is an example of creating a SASReportElement and adding it to the DOM:

```
const myReport = new vaReportComponents.SASReportElement();
myReport.url = ”https://my-viya-server.com”;
myReport.reportUri = "/reports/reports/c3c6befb-3981-4c9e-b011-7dc11dec5e";
myReport.authenticationType = “guest”;
// Set the css classname to be styled by custom css
myReport.className = “custom-report”;
// Set inline styles
myReport.style.width = “100%”;
// Add the report to the end of the body
document.body.appendChild(myReport);
```

As you can see from the example above, in addition to the custom element properties that match the custom tag attributes, you also have access to the built-in element properties such as className and style, amongst others.

INTEGRATING YOUR PAGE WITH REPORT CONTENT

You have now seen how to use both the HTML and JavaScript API to add report content into your web page. This section will show you how to use data-driven content and report parameters as ways to interact with the report from your own code.

DATA-DRIVEN CONTENT

The data-driven content report object is a way to add your own JavaScript visualizations into SAS Visual Analytics reports. This is achieved by embedding your JavaScript inside an iFrame in the report and using window.postMessage() for communication and data passing. Using this form of data-driven content will continue to work using the SDK and requires no special handling. Assuming the objectName of a data-driven content is “ve21”, it could be embedded using a SASReportObjectElement just like any other object:

```
const ddc= new vaReportComponents.SASReportObjectElement();
ddc.url = ”https://my-viya-server.com”;
ddc.reportUri = "/reports/reports/c3c6befb-3981-4c9e-b011-7dc11dec5e";
document.body.appendChild(ddc);
```

Alternatively, the SDK has data-driven content API that allows you to take control of the object from your code. The SDK manages the data and interactions with the other report objects, but it leaves handling of the messages to you. The original data-driven content source that would have been executed in an iframe is no longer used. Instead, the data update messages are routed to you through the SDK API. Data-driven content message callbacks are registered by passing a callback function to the registerDataDrivenContent API, as follows:

```
const ddcHandle = new vaReportComponents.registerDataDrivenContent({
  url: ”https://my-viya-server.com”,
  reportUri: “"/reports/reports/c3c6befb-3981-4c9e-b011-7dc11dec5e",
}, handleDataMessage);

Function handleDataMessage(message) {
  const data = message.data;
  // Do something with the data-driven content data
}
```

This opens up the opportunity to use data-driven content as a mechanism to extract data, calculations, and interactions from SAS Visual Analytics reports and use them to drive your own code. This could be as simple as driving your own custom data visualizations with data from SAS Visual Analytics, or using the user selections from SAS Visual Analytics content to drive external processes.

REPORT PARAMETERS

Report parameters are a SAS Visual Analytics feature that allow you to control the behavior of a report through a set of defined numeric, character, and date parameters. These parameters can control filters, ranks, display rules, prompts, and even calculated columns. This feature can be combined with the SDK to allow your code to manipulate a report and drive interactions, making the SDK more integrated into your page.

The next example will walk through creating a parameterized report that controls a report filter, and then setting that parameter on the SDK through the JavaScript API.

Start by creating a simple report in SAS Visual Analytics, loading the sample data source CARSSASHELP. The report will have a single page, with just two objects; a bar chart showing Horsepower (average) by Type and a pie chart showing Horsepower (average) by Drive Train.

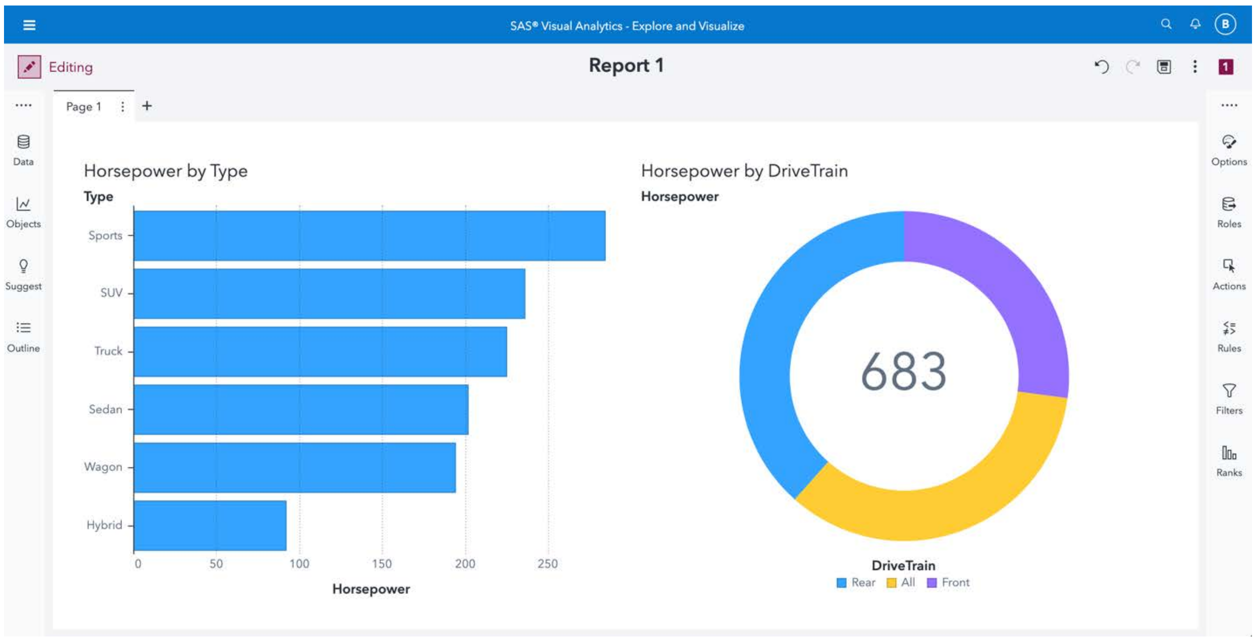
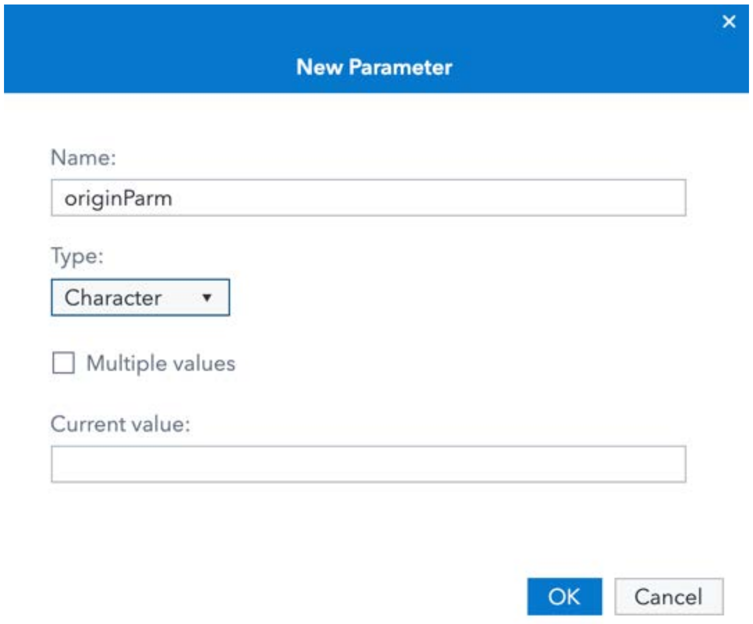


Figure 7: New Report with Bar Chart and Pie Chart

You will also need to create a new character parameter called ‘originParm’. Then create a filter used for both the bar chart and the pie chart that filters where Origin equals the originParm parameter.


New Parameter

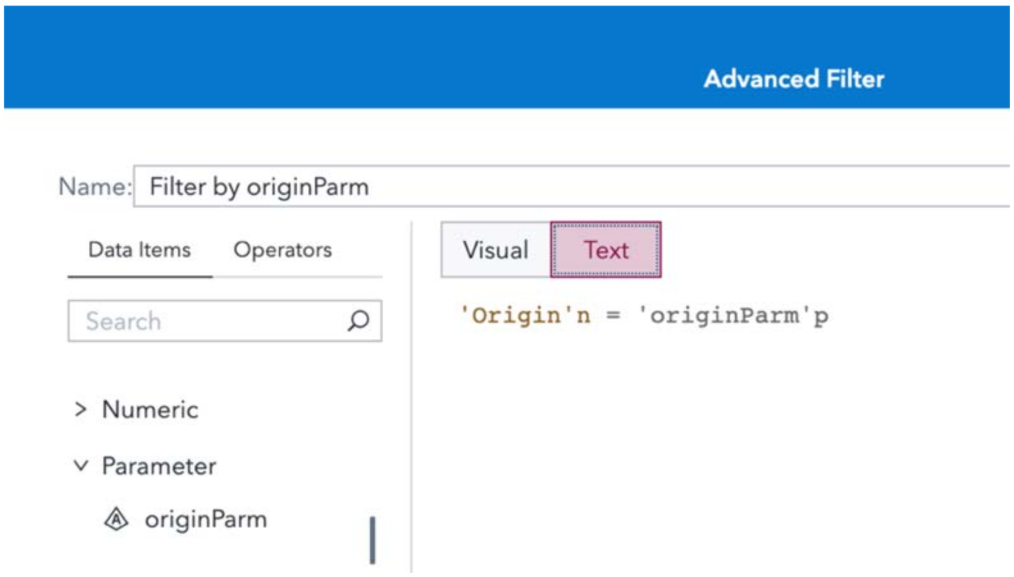
Name: originParm

Type: Character

☐ Multiple values

Current value:

Figure 8: Create a New Parameter



Filter by originParm

```
'Origin'n = 'originParm'p
```

Figure 9: Filter by the Parameter

Now you can use this new ‘originParm’ parameter to control the report filter from the SDK. The report can be included in the page using the sas-report element as shown previously, but to make it easy to reference from JavaScript we are going to assign it the ID “myReport”:

```
<sas-report
  url=”https://my-viya-server.com”
  reportUri="/reports/reports/c3c6befb-3981-4c9e-b011-7dc11dec5e37"
  id=”myReport”
</sas-report>
```

Somewhere in your JavaScript, add the code to set the parameter to “USA”. In this example, our parameter is being used to filter the “Origin” data column, which has values “USA”, “Asia” and “Europe”:

```
const report = document.getElementById('myReport');
report.parameters = {
  originParm: 'USA'
};
```

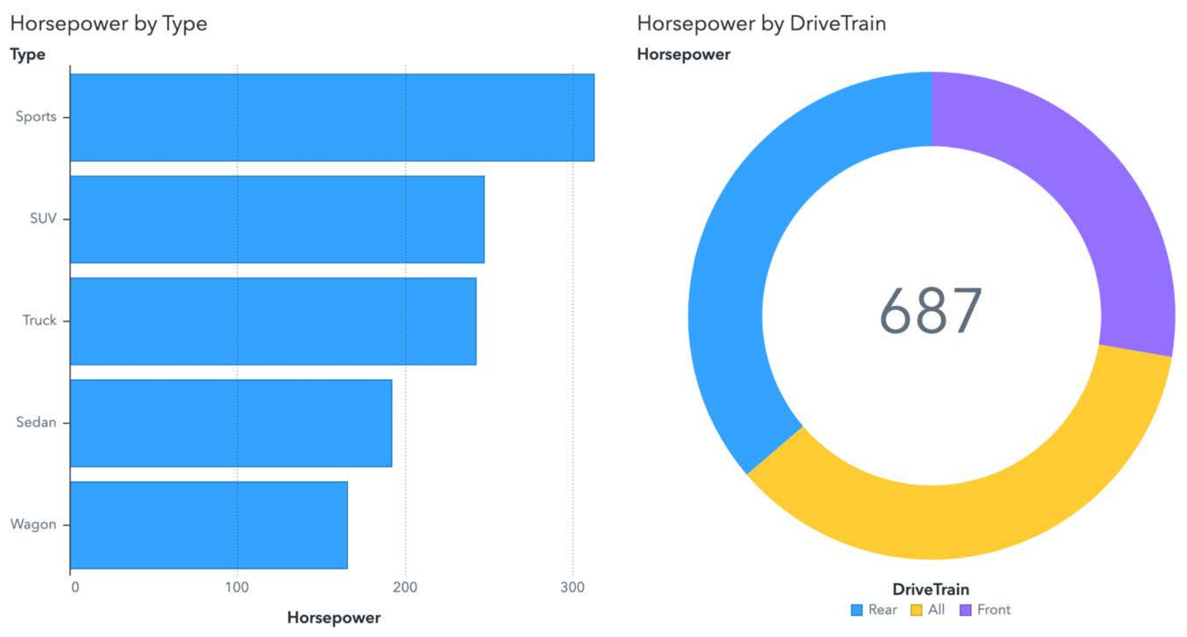


Figure 10: Report Filtered by Origin = 'USA'

CONCLUSION

The SAS Visual Analytics SDK allows you to extend the reach of insights you discover in SAS by embedding them into the web locations where your users already spend their time. By providing both no-code and coding approaches, the SAS Visual Analytics SDK provides a way to quickly publish embedded content, as well as providing capabilities to build dynamic web experiences.

SDKs are drivers of innovation. The customer feedback we have received so far support our belief that the SAS Visual Analytics SDK will open an increasing number of innovative integration opportunities tying together insights from SAS with other systems.

Please share with us how you are using the SAS Visual Analytics SDK and let us know more things you would like to be able to do.

RECOMMENDED READING

- *SAS Visual Analytics SDK developer guide*
- *SAS Visual Analytics SDK Github page*
- *https://developer.sas.com/*

CONTACT INFORMATION

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

Bradley Morris
SAS Institute
919-531-0287
Bradley.Morris@sas.com
Twitter: @_brad_morris_
LinkedIn: linkedin.com/in/brmorr/

Robby Powell
SAS Institute
919-531-8890
Robby.Powell@sas.com
Twitter: @robbypowell
LinkedIn: linkedin.com/in/robbypowell/